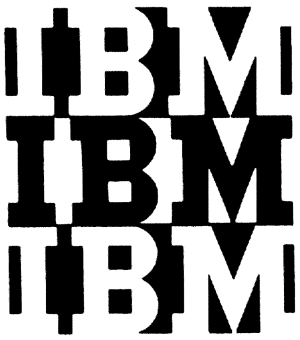


File No. S360-31
GY28-2031-3

Program Logic

Version 8.1

IBM System/360 Time Sharing System

Dynamic Loader

Follows the organization of the dynamic loader in the descriptions of the loading, unloading, and special service routines. Library maintenance, a housekeeping function not part of the dynamic loader proper, is also discussed.

This manual is directed to persons involved in program maintenance, and system programmers who are altering the program design. It can be used to locate specific areas of the program, and it enables the reader to relate these areas to the corresponding program listings. Program logic information is not necessary for program operation and use.

The dynamic loader assigns virtual storage for a task's program modules and resolves address constants for those pages referred to at execution time. In addition, the dynamic loader deletes modules from the task and performs several housekeeping functions.

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

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

IBM System/360 Time Sharing System: Assembler Language, GC28-2000

Fourth Edition (September 1971)

This is a revision of, and makes obsolete, GY28-2031-2 and Technical Newsletter GN28-3128. This edition contains the following changes:

- TSS/360 now supports Q-type address constants (Q-cons) and dummy external definition symbols (DXDs), which allow the user to define the offsets of variables from the beginning of a table. To support this function, two new routines are added to the loader module: Q-CHAIN and RESOLVE Q-REF. In addition, changes are made to EXPLICIT LINKING, ALLOCATE MODULE, DELETE MODULE, and the program module dictionary.

- Extensive changes to DELETE MODULE, as well as changes to GET STORAGE and LOADER LOGOFF, improve the handling of packed control sections that are shared.

- When the dynamic loader encounters a control-section or entry-point rejection condition, it checks REJMSG, a new default value, to determine whether to issue the rejection message.

- A new routine, SETPAGE, has been added to accept and stack requests to build external page table entries. SETPAGE will call the supervisor routine SETXP to have external page table entries built for contiguous virtual storage pages.

- Miscellaneous corrections and improvements are made to the manual.

Each change to the manual is indicated by a vertical line in the margin to the left of the change.

This edition is current with Version 8, Modification 1, of the IBM System/360 Time Sharing System (TSS/360), and remains in effect for all subsequent versions or modifications of TSS/360 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.

This manual describes the internal logic of the dynamic loader of IBM System/360 Time Sharing System (TSS/360). It is intended for persons involved in program maintenance, and system programmers who are altering the program design.

Each of the loader's 36 routines is described and flowcharted. The manual follows the organization of the loader: four sections cover the four modules of the dynamic loader and the subordinate routines that each calls. Three autonomous routines, which are entry points in the main modules, are discussed in separate sections.

An introductory section defines terms, provides an overview of the loader's operations, and concludes with a set of decision tables which permit the reader to trace any possible path through the loader's routines.

Flowcharts for all routines are grouped alphabetically in Section 9.

Appendixes contain additional reference material.

Readers should have a thorough knowledge of TSS/360 assembler language and a general knowledge of the entire system as described in IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003.

PREREQUISITE PUBLICATIONS

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

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

IBM System/360 Time Sharing System: Assembler Language, GC28-2000

CONTENTS

ILLUSTRATIONS

The dynamic loader operates as a set of privileged reenterable system service routines within the virtual storage environment of IBM System/360 Time Sharing System (TSS/360). The loader's function is to allocate virtual storage for user-selected object modules residing in external storage. Realizing that partitioned data sets may be other than object modules (that is, data that is not executable), the dynamic loader ascertains, during the loading process, whether or not the found member is actually a module. If the member is determined not to be a module, the member is rejected as invalid. Included among the loader's routines is the unloader, whose function it is to remove user-selected object modules from the user's virtual storage. Both loading and unloading are performed in response to explicit user request. The loader is dynamic in the sense that only address constants on text pages actually referred to at execution time are resolved by the loader. The auxiliary functions of loader logoff and library maintenance are also included in the set of loader routines.

Program Modules

The program module is the primary output of all TSS/360 language processors. The input to a language processor (that is, the FORTRAN compiler, the PL/I compiler, or the assembler) consists of a stream of source language statements that are translated into hexadecimal instructions and data. Input to the linkage editor, a service program, consists of directive statements and a set of object modules to be combined into one output object module. The object module, a member of a partitioned data set, is divided into three parts: the program module dictionary (PMD), one or more control sections, and, optionally, the internal symbol dictionary (ISD). The PMD and ISD contain information used by system programs; the control sections contain the "program" -- the hexadecimal instructions and data that are the translation of the user's source language statements.

Control Sections

The control section organization is determined solely by the user in the case of machine language assembly, by the user and language processor in the case of the FORTRAN compiler, and solely by the language processor in the case of the PL/I compiler. Within the PMD (described in Appendix B) are the control section dictionaries (CSDs), which describe each control section in the module. There is one CSD for each control section and, indeed, aside from a small amount of header information, the PMD consists mainly of a group of CSDs.

The language processors assign text locations within each control section relative to the base of the control section. This allows the dynamic loader to allocate virtual storage independently for each control section within a module. The dynamic loader operates on object modules as its gross building blocks, but allocates virtual storage by control section group. (A control section group consists of all those fixed-length control sections in a single module having identical attributes.) If the control section packing option is not used, storage is allocated to the control section group beginning on a page boundary and for an integral number of pages; if control section packing is used, storage is allocated to the control section group beginning on a doubleword boundary. Variable-length control sections are allocated storage beginning on a page boundary, for an integral number of pages.

At the time the user creates a control section, he may assign to it one or more of eight attributes:

1. Fixed-length control sections -- A fixed-length control section is allocated an integral number of pages of virtual storage by the loader. This number of pages is the minimum that will contain the limits of the control section. For example, in an assembly of CSECT A, if the location counter stops at (decimal) 11,000, the loader allocates three pages of virtual storage at load time. Fixed-length is the default attribute when variable-length is not set.

2. Variable-length control sections -- A variable-length control section is allocated pages in addition to the required minimum (defined above). This additional number of pages is an installation parameter. (If the variable-length attribute bit is not set in the CSD, the control section is fixed-length by default.)

3. Read-only control sections -- Read-only control sections are allocated storage with a protection key that prevents the user from storing or

writing into any byte of the control section.

4. Privileged control sections -- Privileged control sections are allocated storage with a protection key that disallows reference to the control section by any other than a privileged system service routine. An attempt by a nonprivileged user to write or read a privileged control section results in a storage protection error. Privileged control sections contain only entry points whose names begin with CHB or CZ. The normal user may not declare privileged control sections; the loader will erase such an attribute from control sections in any module except those that come from the system library (SYSLIB).

5. System control sections -- Control sections marked "system" are maintained by the loader so as to prohibit user reference to them, except through SYS symbols. 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.)

   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 but those that are loaded from SYSLIB.

   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.)

6. Public control sections -- Public control sections are assigned storage by the loader in such a way as to make the control section available to more than one task at the same time. For example, if two or more tasks each reference a public control section named A, and that reference is resolved from the same shared data set, they all share the same physical copy of A. The first task to reference A will cause the loader to allocate A to public (known as shared) 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. An attempt is made 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) for external references.

7. Prototype control sections -- Prototype control sections (PSECTs) are allocated storage such that they are packed on page boundaries within a segment. (The loader ordinarily makes no such effort to pack nonprototype control sections from different modules within a segment.) PSECTs will normally contain the private copy of modifiable storage that is made available to each task for public routines, the executable control sections of which have been allocated public storage. This modifiable storage will consist of save areas, working storage, and variable program data. The normal reenterable module will consist of a public control section containing executable instructions and a prototype control section containing all the adcons and other modifiable data.

8. Common control sections -- Any language processors may produce control sections with the common attribute. The loader examines the common attribute only in the event of control section rejection and then only for diagnostic purposes (see Appendix E, "Loader Diagnostics").

Module Residence -- External

To be available to the user, each object module must be contained in one of the program libraries. Furthermore, the library that contains a user-required module must be made accessible by the task. There are three types of program libraries in the TSS/360 environment:

1. The system library (SYSLIB) is the source of all standard system routines and is available to all users.

2. The user library (USERLIB) is a private library assigned to each user when he joins the system. This library is associated with the user's ID and is made available to him at LOGON.

3. The job library (JOBLIB) is a library that the user defines by means of DDEF commands. The user is allowed to define any number of JOBLIBs during his task, and these are normally used for the purpose of stowing away and retrieving object program modules created by the language processors.

Thus, SYSLIB and USERLIB are automatically accessible to each task and to the dynamic loader, as are those job libraries that the user defines during his task. These libraries form a hierarchy for searching purposes:

1. Job libraries
2. USERLIB
3. SYSLIB

The job libraries are at the lowest position in the hierarchy; while SYSLIB is at the highest position (that is, it is searched last). Within the job libraries, a hierarchy is dictated by the order in which the job libraries were defined; the last-defined library will be searched first. The user may review the job library hierarchy by the DDNAME? command, and may move a job library to the top of the library list with the JOBLIBS command.

Each of these libraries must be a partitioned data set. Each program module, then, is a member of the partitioned data set, while each entry point and non-common control section name is an alias for that member. Thus, a module may be loaded by module (that is, member) name, or by any alias name. Object program libraries created during the process of assembling, compiling, or link-editing are automatically formed as partitioned data sets. The user's only responsibility is a DDEF command, with a JOBLIB option, for each job library that he wants to establish for his task.

The actual dsname of the system library is SYSLIB; of the user library, USERLIB, while the job libraries are named by the user -- the word JOBLIB being a keyword operand value in the DDEF command. Libraries defined by DDEF commands that omit the JOBLIB keyword are not accessible by the loader.

The purpose of the library hierarchy is to allow supersedure of routines by hierarchy position. For example, if more than one open library contains a symbol name that is the object of search by the loader, the symbol contained in the lowest library in the hierarchy will be extracted.

Module Residence -- Internal

Once it has been determined that a module is to be loaded from an external library, it is allocated space within the user's virtual storage. The dynamic loader deals only with the module's PMD and text; the module's ISD, if any, is not examined or processed in any way. The loader transfers the PMD from the partitioned data set into a chain of PMDs known collectively as the task dictionary (TDY). The text pages of the module are allocated virtual storage addresses; however, the loader itself does not transfer the text pages. The resident supervisor paging mechanism causes a text page to be transferred physically to real storage the first time the text page is referenced by the user's code. Unreferenced text pages are never transferred into real storage. See Figure 1 for a pictorial summary of module structure and residence.

External Symbol Definitions

External symbol definitions (DEFs) are symbols within a module, referable by name from other modules. Referability is effected by the language processor's creation of DEF tables within the control section dictionary (CSD) which name all symbols referable by other modules. DEFs arise from three sources:

1. Creating a module causes a DEF entry to be created for the module name, alternatively referred to as the standard entry point.

2. Declaring a control section (including common) causes a DEF entry to be created whose name is the name of the control section. If blank common is declared, the name is a name of blanks. If an unnamed CSECT is declared, its name is the module sequence number.

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

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

Absolute DEFs are those whose names are defined by an EQU statement with an operand that is an absolute expression. For example, the code:

```
        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 at execution time.

Relocatable DEFs are those whose value is storage allocation dependent. For these DEFs, the language processors always output a value relative to the base of the defining control section. For example, if CHXAAA is the label of some statement at

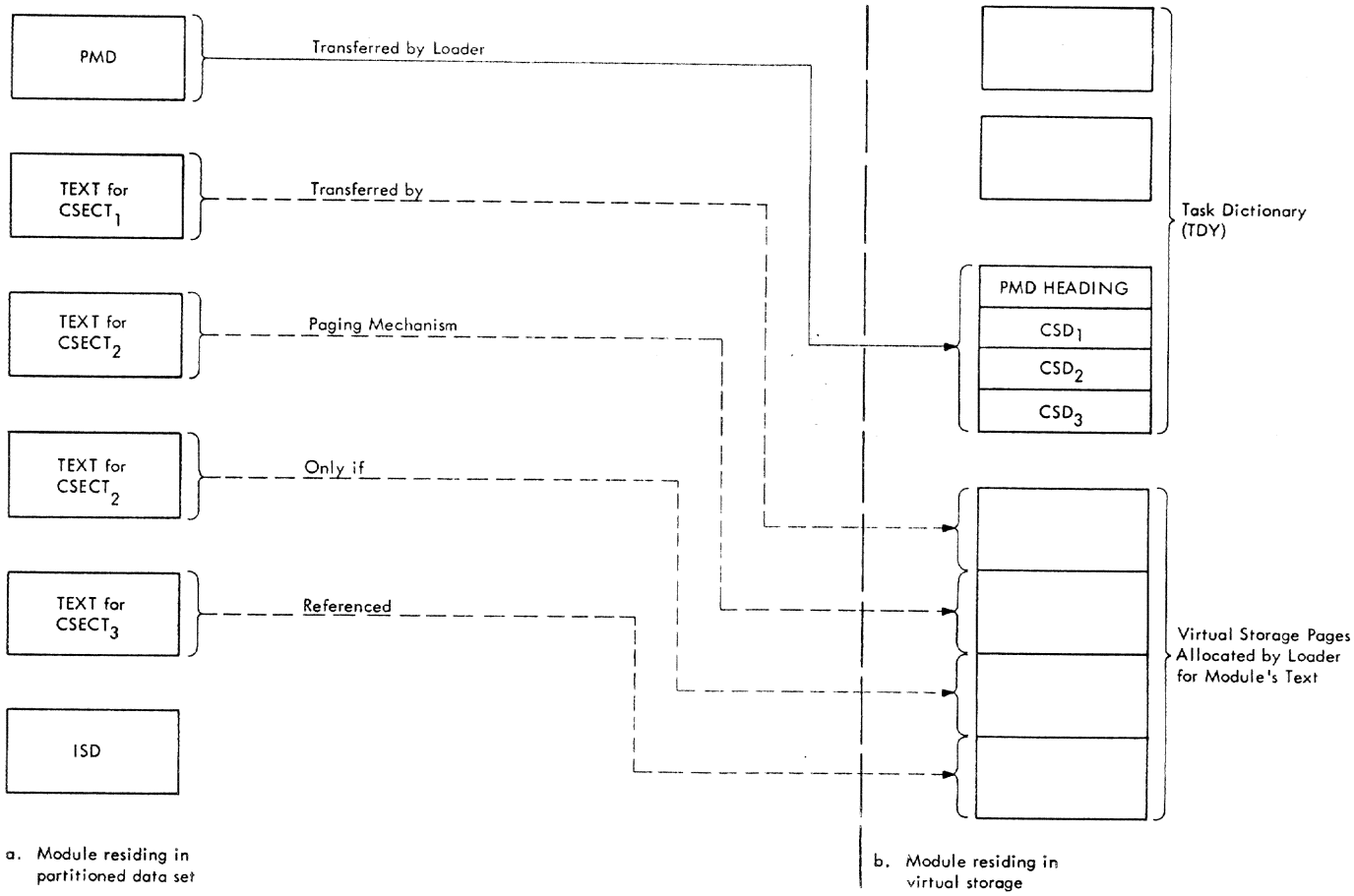


Figure 1. Program Module Structure and Residence

byte location 1000 relative to the origin of its control section, the code

```
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 address of the defining control section as allocated by the loader.

Complex DEFs are of two types. Type 1 are simply relocatable DEFs whose ENTRY statement appears 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 created by the language processor. This is effected by an external reference (REF) which names the control section containing the definition. Type 2 are DEFs of which the symbol is defined by an EQU statement whose operand field contains one or more external symbols.

There is a special entry point used to define the nominal execution starting point for a module: the standard entry point. The language processor prepares a complex DEF entry for this entry point whose name is the module name. This treatment allows external references to modules by name; that is, a V-con or R-con naming a module is a legal coding practice.

External Symbol Values

TSS/360 has adopted a convention for linkage to reenterable routines, requiring that any external symbol have associated with it two values:

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

2. The R-value, which is the virutal 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 that instruction in the public, reenterable CSECT that the user wants to be executed first at routine entry. The R-value is used to identify the origin of the related private PSECT. The user effects this by writing in the PSECT an ENTRY statement naming the label of the first instruction in the CSECT to be executed. 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 will be used to demonstrate:

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

Assuming further that the loader assigned PSECT A to virtual storage location X'800000' and CSECT C to virtual storage location X'205000', these are the V-values and R-values 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

*DEF for control section name

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 control section name is generated by the control section statement -- not by an ENTRY statement. Therefore, the control section name DEF entry is always in the CSD for that control section such that the DEF for a control section name always has an R-value equal to the V-value. (Note A and C, above.) Given these definitions, then, a reenterable module should not be called by a control section name.

The program module name, or standard entry point DEF, is treated somewhat differently. Its V-value is the value of the expression contained in the END statement of an assembly; the language processor prepares a complex DEF entry for the standard entry point from the END card 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, the V-value for the standard entry point will be 212004. The R-value for the standard entry point is computed at load time by the dynamic loader. 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. Furthermore, the module name then assumes all the attributes of the PSECT or CSECT that defines the R-value.

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

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

In the case of a blank END statement or compiler output, the "value of the expression contained in the END statement" may be taken to mean the address of the first executable statement in the compilation.

External Symbol References

External symbol references, or REFs, are symbols referred to within a module but defined outside the module. The user can create a REF 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 control section containing the statement. The symbols named in V-con and R-con statements will also generate REF entries.

REFs may also arise due to complex DEFs. For example, if CSECT AA has some symbol, X, and PSECT BB has the statement

```
ENTRY   X
```

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

There will also be at least one REF for the module name complex DEF. For example, the statement

```
END     Z
```

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

External references to other modules provide the means by which the loader links modules implicitly; that is, in a module the occurrence of a REF that names a symbol not defined in the module will provoke a search to locate that module that contains some DEF entry whose name matches the REF name.

External Dummy Sections

External dummy sections facilitate communication between programs by allowing the user to define work areas in several different programs and then at execution to combine them into one block of storage accessible to each program. Several different programs can be assembled together, each with one or more external dummy sections; after the loader processes these programs, the user can allocate storage for the dummy sections in one block. An external dummy section is defined through the use of a DXD instruction, or a DSECT, in combination with a Q-type DC instruction. In order to allocate the correct amount of storage when the program is executed, the user must use the CXD instruction in at least one of the programs.

User Authority

Users of TSS/360 are divided into three authority classes. A code, identifying the class, will be assigned to each user in accordance with the user's programming assignment and requirements. This assignment, made when the user is joined, is maintained on file by TSS/360. When the user "logs on" the system, this authority code is extracted from the file and associated with his task. The user himself never has any program control over his authority code. The three authority codes and classifications are:

1. Code U -- Referred to in this document as the "normal user." This is the classification normally assigned to the applications programmer not concerned with system maintenance.

2. Code P -- This identifies the "system programmer" who is concerned with a limited level of system maintenance.

3. Code O -- This identifies the privileged system programmer or system operator, both of whom are concerned with the highest level of system maintenance.

The loader makes use of the authority code in a variety of ways. In general, the loader uses the authority code to determine the level of system "protection" that it will enforce in loading modules. The loader imposes total system protection for the normal user, and progressively less protection for the two classes of system programmers. Specifically, the loader uses the authority code to determine in which hash table to search for or post symbols (see "Split Hash Table"). The loader also may alter control section attributes according to user authority and judge on the admissibility of certain forms of symbols when loading modules. The total impact of authority codes on the loader's processing of symbols and control sections is summarized in Appendix A.

Split Hash Table

The loader employs a common symbol randomizing (hashing) technique to speed symbol lookup during the loading and unloading process. This technique involves the use of three separate tables that contain the origin of the "hash chains" of DEF entries. Two of the tables are used for system symbols (that is, symbols found in routines extracted from SYSLIB whose control sections are marked system). One system table (SYSHASHP) is for symbols from privileged control sections, the other (SYSHASHNP) for symbols from control sections not marked privileged (that is, nonprivileged). The nonprivileged system table contains symbols posted as a result of loading the assembler, FORTRAN compiler, PL/I compiler, or linkage editor. When user authority is P or O, only these two tables are constructed.

In addition to the system tables, a third hash table is constructed for the normal user (authority U). The use of this additional table gives rise to the term "split hash." It is employed to provide close control over the interface between the normal user and system routines. This control is effected by separating the normal user's symbols from system symbols, thus obviating erroneous linkage from system routines to normal user routines and vice versa. The normal user's symbols are

placed into the user hash table (USERHASH); symbols from system control sections are placed into the appropriate system hash table. When modules are loaded and linked together, the loader obeys the general rule that REFs from system control sections are satisfied by DEFs in the system tables, while REFs from control sections not marked system are satisfied by DEFs in the USERHASH table. An exception is when a REF from a nonsystem control section begins with SYS, in which case it can only be satisfied by a DEF SYSHASHNP. This symbol resolution technique is also summarized in Appendix A.

Load Errors

Whenever the loader encounters an anomalous situation in its processing, it advises the user by means of a message on SYSOUT. All such diagnostic messages are listed in Appendix E. Certain serious error conditions warrant the loader's making additional load error indications, either to the task monitor or directly to the user. Such conditions are those that are liable to impair further execution of the task (such as the loader's being unable to locate the module named on a LOAD statement).

Control over this additional load error indication is in the hands of the user in the case of the LOAD macro instruction. The C2 digit within the LOAD adcon group can be set by the user. Normally, this digit is set to zero; should a serious error occur, the user is given the appropriate diagnostic by the loader, after which control is returned with an error indication to the task monitor.

If the user should set the C2 digit to one prior to executing a LOAD or CALL macro instruction, and the loader should detect a serious error, the C2 digit is set to seven as the error indication to the calling program (which may initiate program checks for such condition). In this case, the loader returns to the task monitor without error indication, and the user is not prompted by the command analyzer and executor.

MAJOR FUNCTIONS OF THE DYNAMIC LOADER

The collection of routines known as the dynamic loader performs these privileged system service functions:

EXPLICIT LINKING PAGE RELOCATION	These two make up the loading processors.
EXPLICIT UNLINKING	This is the unloader.
LOADER LOGOFF LOADER CLEANUP	These two special service routines are called by the command system logoff processor.
LOADER RELEASE	This is a special service routine called by the command system release routine.
LIBE MAINT	A special program library service routine.

Explicit Linking

EXPLICIT LINKING is called in response to a LOAD or CALL macro instruction or to a LOAD or RUN command. The major argument to EXPLICIT LINKING is the name of some symbol. The symbol can be a module name, a control section name (other than common), or any other module entry point. If the module defining the argument symbol is not already loaded, the loader loads the module's PMD into the task directory, allocates virtual storage for the module's control sections, performs a variety of diagnostic checks, loads additional modules to satisfy external symbol references, supplies values for CXD instructions, and sets up segment, page, and external page table entries for the text pages of each allocated control section.

At this point, the EXPLICIT LINKING phase of the loading process is complete and virtual storage has been allocated. None of the control sections' text has yet been actually moved into real storage. Instead, each page of text for which storage was allocated is listed in the page tables as "unavailable." Furthermore, the loader has identified those pages of text containing relocatable adcons and additionally marked these pages additionally as "unprocessed by loader." This latter marking is placed in the external page table.

Page Relocation

When the task subsequently makes a virtual storage address reference to a page that is on an "unavailable page," an interruption occurs. The resident supervisor responds to this interruption and, through its paging mechanism, transfers the referenced page from its partitioned data set residence into real storage. If it is also noted that the page in question is unprocessed by loader, the resident supervisor calls the task monitor, which effects immediate linkage to the PAGE RELOCATION entrance to the loader.

PAGE RELOCATION is the name given to the process of resolving the adcons in a page

page of text in virtual storage. This function is performed in a given task only once for each text page with adcons. It occurs at the first "page unavailable" interruption; that is, when the text page is first brought into real storage.

The resolving of adcons always involves the application of some REF value to the text bytes to be resolved. All such REF values were already computed during the EXPLICIT LINKING phase of the loading process.

Explicit Unlinking

EXPLICIT UNLINKING is called in response to a DELETE macro instruction or UNLOAD command. The major argument to EXPLICIT UNLINKING is the name of some symbol. The symbol can be a module name, any control section name, or any other module entry point. If the argument symbol is defined in the task, the containing module is unloaded from the user's virtual storage, provided the module was explicitly loaded in the first place, and provided there are no outstanding external symbol references from other modules in the task defined by DEFs within the subject module. Unloading consists of the removal of the unloaded module's PMD from the task dictionary and the freeing of virtual storage for all the unloaded control sections.

Loader Logoff

LOADER LOGOFF is called at task end by the logoff processor for terminal housekeeping. It is called only once per task, and its function is to adjust the shared data set table (SDST) to show that the current task is no longer a shared user. User counts for each of the task's public control sections are decremented in the SDST. If counts are nonzero after adjustment, only the current task is disconnected from the shared page tables. If the user counts go to zero after adjustment (a task might be the only one currently using the shared storage), the public storage is released through FREEMAIN. Private storage is not released by LOADER LOGOFF; the private page tables for the task will be eliminated at the time the task's TSI is deleted. This deletion follows LOADER LOGOFF's processing.

Loader Cleanup

LOADER CLEANUP is called by the command system logoff processor at the end of each sub-task during express batch processing. LOADER CLEANUP calls LOADER RELEASE to unload from virtual storage all those modules which were loaded for the specific sub-task, so that only IVM modules remain in virutal storage for the next sub-task.

Loader Release

When a 'DDEF' for a job library is released, the release command routine calls LOADER RELEASE to unload any non-IVM modules loaded from that library. LOADER RELEASE determines which modules should be unloaded, then calls EXPLICIT UNLINKAGE to do the unloading. If any module cannot be unloaded because of outstanding references, a message to the user is printed and an error return code set for the release command routine.

Library Maintenance

The LIBE MAINT routine is called to maintain the data control block (DCBs) for the program library list; that is, the hierarchy of open partitioned data sets in a task available for access by the dynamic loader. It is called at the beginning of each task to open the system library (SYSLIB) and the user's private library (USERLIB). Furthermore, LIBE MAINT will be called at times during the life of a task in response to any JOBLIB DDEF command entered by the user.

The dynamic loader has no direct interface with LIBE MAINT. Instead, LIBE MAINT is called by other system routines to open and close DCBs for the user's data sets. The open DCBs are chained together, and the loader will make use of this chain in attempting to locate symbols during EXPLICIT LINKING.

THE LOADING PROCESS

Invoking the Loader

The user may initiate the loading process by a LOAD or RUN command. This action causes the command analyzer and executor to issue a LOAD macro instruction to which the dynamic loader responds by loading the named module. The user may also write inline statements in assembler macro language to invoke the loading process. These will take the form of the LOAD or E-type CALL macro instruction. The LOAD macro instruction is used only to effect the loading of the named module; CALL causes both loading of and branching to the named module.

When CALL and LOAD macro instructions are expanded the following code is generated:

At the point of CALL		At the point of LOAD	
DS	0H	LA	15,CHD&SYSNDX
L	15,CHD&SYSNDX+12	EX	0,0(0,15)
L	14,CHD&SYSNDX+16		
ST	14,72(0,13)		
BASR	14,15		

while in the user's first declared PSECT the adcon group is generated as follows:

```
            CNOP    0,4
CHD&SYSNDX  SVC     127         SVC for explicit loading
            DC      H'C1C2'     Option codes
            DC      CL8'name'   Module name (or alias) of module to be loaded
            DC      A(*-12)     V-value of name filled in here by loader
            DC      F           R-value of name filled in here by loader
```

When the DLINK SVC is executed, the task monitor takes control and effects a type-I implicit linkage to the dynamic loader with GR1 pointing to a fullword that contains the virtual storage address of the adcon group. The loader proceeds to allocate virtual storage for the named module and to place the V- and R-value in the adcon group, loading any and all modules required to resolve external symbols.

When the loader has completed these actions, it returns to the task monitor. In the case of error-free processing, the task monitor determines whether a CALL or LOAD was executed. (The loader's return code indicates both error condition and type of adcon group; that is, CALL or LOAD.)

In the case of a LOAD, the task monitor merely returns control to the point in virtual storage immediately following the EX instruction that occasioned the DLINK. In the case of a CALL, the task monitor picks up the resolved R-con and places it in the 19th word of the calling program's save area; that is, in the 19th word following the address contained in register 13. The task monitor effects linkage by placing the resolved V-value into the IC field of the user's old PSW, so that the next time this PSW is fetched, the called routine is entered at the V-value location. At the point of entry, register 13 will be pointing to the caller's save area, the 19th word of which will contain the R-value, by convention the PSECT origin of the called routine.

In the case of an error return from the loader while in conversational mode, the task monitor will effect linkage to the command analyzer and executor to prompt the user. The task monitor disregards error codes in the nonconversational mode.

The option codes, C1 and C2, are interpreted by the loader as follows:

C1 Code: A CALL is indicated when the low-order bit is a 1. A LOAD is indicated when the low-order bit is a 0. If the high-order bit of the C1 byte is set in an adcon group located in a system control section, the loader will resolve the adcon group symbol from the USERHASH table rather than from one of the system tables, from which such system adcon groups are normally resolved when the bit is not set. This feature is implemented so that the load command processor, which is a system routine, may make explicit LOADs of user routines in response to the LOAD command.

C2 Code: The C2 digit governs the loader's actions in the case of serious load errors encountered during the response to a LOAD macro instruction. The details of these actions are described under "Load Errors."

Allocation

The loading process is divided into two phases: the virtual storage allocation (explicit linking) phase and the text page relocation (adcon resolution) phase. The allocation phase commences in response to the execution of a LOAD or CALL macro instruction; text page adcon resolution is effected as the page to be relocated is referenced by the user's code.

Symbol Lookup

Allocation begins with the looking up of the symbol name to be loaded. The appropriate hash chain in the TDY is searched first. If the loader finds the symbol defined there, no allocation is necessary, since the module defining the symbol is already a part of the task and has had virutal storage allocated. The loader merely fills in the V-value and R-value in the adcon group associated with the calling sequence, and returns to the user via the task monitor. If the symbol cannot be found in the TDY, a library search is initiated. If the symbol is not found, an error condition exists, that is, the symbol is undefinable for this task.

PMD Loading

If a library is found that defines the symbol name, the defining module's PMD is transferred from the partitioned data set into the TDY, maintained in each task's virtual storage. The count of modules loaded from this library, found in the TDTBLK portion of the JFCB, is incremented by 1.

Control Section Rejection

After the PMD is loaded into the TDY, each control section name within the module is checked. Those control sections whose names either duplicate entry point names already within the TDY or whose names are determined illegal (see Appendix E, "Loader Restrictions") are rejected. This process of control section rejection has the side effect that none of the entry points defined by the control section will be entered into the appropriate TDY hash chain, meaning that references to these entry points must be satisfied elsewhere or not at all. Control section rejection finds its primary application in the treatment of common control sections. The loader will accept the first common control section it encounters of a given name (or blank), reject all subsequent common control sections of the same name (or blank), and tie all common references to the loaded common control sections. Sometimes control section rejection may be accompanied by or caused by the anomalous conditions summarized under "Loader Restrictions."

The treatment of unnamed control sections deserves some special comment here. Unnamed common control sections are assigned a name of eight alphameric blanks. After the first common control section is loaded, subsequent unnamed common control sections will be rejected, as discussed above.

Unnamed CSECTs are assigned a name of 16 hexadecimal zeros by the assembler. To render such names unique to the module in which they were declared, the loader places a module sequence number in the low-order 16 bits of the first word of the name part of the DEF entry and all REFs of the same "zero" name within the module. This technique eliminates control section rejection for unnamed CSECTs, since unnamed CSECTs from different modules will be distinguishable one from another. (See Appendix D, Restriction 12.)

Control Section Storage Assignment

Virtual storage is allocated for each of the nonrejected control sections in the module. Fixed-length control sections of identical attributes within a module are allocated storage as a group. Variable-length control sections are allocated storage individually. (The system actually allocates an additional fixed number of pages in response to the variable-length allocation request.)

Storage protection keys are set up for each control section group at the time storage is requested for that group. Read-only control sections are assigned a storage key that will not allow the user to store in the virtual storage assigned. Privileged control sections are assigned a storage key that will not allow the user to store into or to read the assigned virtual storage. Privileged control sections will only be found in certain system service routines; the user is not allowed to declare such control sections. All other control sections are assigned a storage key that allows unlimited user reading and writing of the assigned storage.

Public control sections in modules loaded from shared data sets are assigned shared storage, to make such control sections potentially available to other tasks. If some public control section has not previously been allocated by some other task, the loader will assign shared storage such that this task's copy of the control section will be loaded into the shared storage. If some public control section has already been allocated shared storage by another task, the current task is merely "connected" to such shared storage; that is, all references to such public storage will be tied to the control section already loaded.

Page table entries are set up for each of the nonrejected control section text pages. The external library storage address is associated with each external page table entry, and each page is marked unavailable. The first user reference to any byte on the page will cause an interruption. This interruption will cause the paging supervisor to transfer the page from the external library into real storage.

At the time the page tables are set up, the loader checks each page for the presence of adcons. Those pages containing adcons are marked unprocessed by the loader in addition to the unavailable marking. The referencing of pages marked unprocessed by loader will cause the paging supervisor to effect a call via the task monitor on the page relocation entrance of the loader. This action is described more fully under "Relocation."

DEF and REF Processing

The value of all DEFs in the nonrejected control sections of the module are computed except those DEFs whose names duplicate DEFs previously loaded or whose names are judged illegal. Duplicate or illegal DEFs are rejected with diagnostics. Relocatable DEFs are computed by adding to the DEF value the virtual storage base address allocated by the loader to the containing control section. Absolute DEFs require no computation. Complex DEFs are computed last. Recall that complex DEFs have associated with them REFs to other control sec-

tions. If the external name to which such a REF refers is not found in the TDY, the entire loading process is initiated to load a module that will so define the REF. After the complex DEFs are computed, all of the remaining REFs in the module are satisfied, which may effect the loading of additional modules. 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 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 by control section rejection. For example, some module has a REF to symbol X which is found in CSECT C in module A in some library. During allocation, CSECT C is rejected by the prior occurrence of some other CSECT C, such that when allocation for module A is completed, symbol X is still unsatisfied. The loader checks for this condition and accommodates it by initiating the symbol search (and allocation cycle) once again, this time in the next library in the hierarchy. A symbol is undefined when all libraries from the hierarchy starting point to and including SYSLIB have been searched, yielding no definition.

Page Relocation

Whenever a page-unavailable interruption occurs, the paging supervisor transfers the page into real storage and checks the unprocessed-by-loader bit in the external page table. If this bit is not set, no loader action is required. If the bit is set, the paging supervisor effects a call, via the task monitor, on the page relocation entrance to the dynamic loader. The loader's action, in this event, is merely to compute the correct virtual storage value of each adcon in the page triggering the interrupt. The processing of adcons will always involve the application of some REF value to that portion of the text occupied by the adcon. There are five possible applications:

1. Add the V-value of an external or internal REF to the text value.
2. Subtract the V-value of an external or internal REF from the text value.
3. Store the R-value of an external or internal REF into the text.
4. Store the value of a Q-REF into the text.
5. Store the value of a CXD-REF into the text.

When this page relocation occurs, all REF values will have been satisfied (during the allocation phase of the loader). Once all adcons have been resolved, the loader returns to the task monitor and eventually to the instruction in execution when the page-unavailable interruption occurred.

Loading Example

Assume that M1, a module already loaded, executes the following statement:

```
CALL  SINE, E
```

The loader will search the TDY looking for some entry point named SINE. Assuming that SINE is not currently loaded, the loader will initiate a library search for SINE. If found, the allocation of the module containing SINE commences. Assume now that SINE is contained in module M2, and that M2 has two REF entries whose names are R1 (satisfied in module M3) and R2 (satisfied in module M4). Once storage is allocated for M2, the two REFs will be processed. R1 will cause the loading of M3 if it is not already loaded, and R2 will cause the loading of M4 if it is not already loaded. Assuming that M3 and M4 have no external REFs, following the allocation for M3 and M4, the V-value and R-value of SINE are filled in the adcon group in M1, thus completing the allocation phase of the loading process. Figure 2 shows the resulting allocation and links between modules.

During the allocation, of course, all text pages were marked unavailable and those with adcons marked unprocessed by

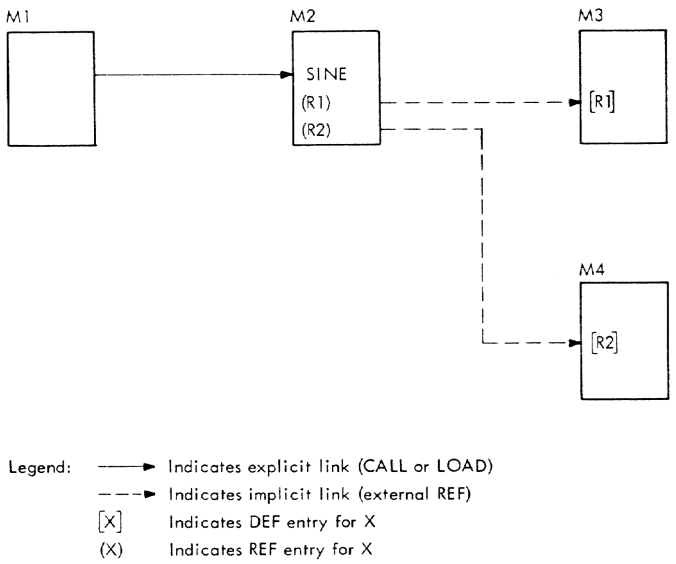


Figure 2. Loading Example

loader. At the conclusion of allocation, control via the task monitor will pass to the entry point SINE in module M2. When the branch to SINE or its V-con is interpreted for execution, a page unavailable interruption probably will occur, and the page containing SINE will be paged into real core. If there are unprocessed adcons on this page, the relocation phase of the loader will be called to compute the correct virtual storage values of these adcons and to store them in the text, which is in storage for the first time.

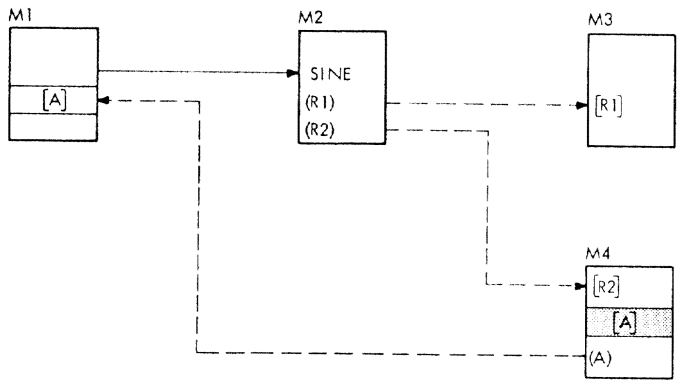
Altering the above example to show an example of control section rejection, assume that M4 refers to some named common A which is defined in M4 and has also been previously loaded as a result of being declared in module M1. Figure 3 is a diagram of this situation. Note that common A in M4 is rejected, and that references to A within M4 are satisfied in module M1's common A.

THE UNLOADING PROCESS

Invoking the Unloader

The user can initiate the unloading process by the terminal command UNLOAD, or by the inline coding statement DELETE. The DELETE macro instruction expands as:

```
EX      0,CHD&SYSNDX
```



Legend:
- ——→ Indicates explicit link (CALL or LOAD)
- - -→ Indicates implicit link (external REF)
- [X] Indicates DEF entry for X
- (X) Indicates REF entry for X
- Indicates rejected Control Section

Figure 3. Loading Example Showing Control Section Rejection

at the point of call, while in the user's first declared PSECT the DELETE adcon group is generated as:

```
         DS      0F
CHD&SYSNDX SVC   123          SVC for unloading
         DC      CL8'name'    Module name (or alias) to be unloaded
         DC      H'C3C4'      Unload options and return code
```

When the Unload SVC is executed, the task monitor takes control and effects a type-I implicit linkage to the unloader, with GR1 pointing to a word that points to the DELETE adcon group. The unloader proceeds to unload the named module and possibly modules referenced by it from the user's virtual storage, according to the option byte, C3. Upon completion of the unloading process, the unloader returns to the task monitor, which then returns to the user's code following the EX instruction.

The first byte of the halfword of options and return code is used to allow two variants to the standard DELETE:

1. The high-order bit of C3 may be set to reverse the effects of the system attribute of the control section containing the DELETE adcon group. (See the discussion of the CALL/LOAD adcon group high-order C1 bit, under "The Loading Process.") This feature is implemented so that the UNLOAD command processor, a system routine, can issue DELETE macro instructions on user modules in response to the UNLOAD terminal command.

2. The low-order bit of C3, if set, directs the loader to unload only the module defined by name in the DELETE adcon group. The unloader makes no attempt in this case to delete modules referenced by the named module.

C4 is used to contain the unloader's return code.

Creating Deletion Candidates

The explicit unlinking entrance to the dynamic loader is called whenever a DELETE macro instruction is executed. The major argument is some symbol, either a module name or alias (control section name or other entry point name) whose containing module is to be unlinked from all other programs and deleted from the task.

Module deletion, or unlinking, includes several processes:

1. Locating all explicit references to the module to be deleted and "rearming" them.

2. Tracing explicit references from this module to identify subordinate modules that may be deleted as well.

3. Tracing implicit references from this module for the same purpose as (2).

4. Deleting all extant DEFs defined in the deletable modules from the DEF chains.

5. Deleting all control sections and freeing allocated storage.

6. Deleting all deletable modules' PMDs from the TDY.

A deletion candidate, then, is either a module whose name (or alias) appears in the DELETE statement (primary candidate), or another module (secondary candidate) that is referenced by the primary or by the other secondary candidate. There are two ways in which a module may reference another module. An explicit reference is effected by a module's executing a LOAD or CALL macro instruction naming an external symbol defined in another module. An implicit reference is effected by a module's having a REF entry that is satisfied by a DEF entry in another module.

The allocation phase of the loading process sets up appropriate explicit and implicit chains linking referenced PMDs with referencing PMDs. Secondary deletion candidates are located during the unloading process by tracing these chains and placing every referenced module on a candidate list. This tracing process cascades until all modules referenced by the primary and secondary deletion candidates have themselves become deletion candidates.

Eliminating Deletion Candidates

Only those deletion candidates that have no outstanding explicit or implicit references to them are retained on the candidate list. The removal of any candidate on the list may result in the removal of a previous candidate from the list. Now this process is reiterated until a stable candidate list results, and all those modules remaining on the list may be deleted from the task.

There is one exception to the forgoing algorithm. The primary deletion candidate is deleted so long as there are no outstanding implicit references to it. Explicit references to the primary candidate are traced to their source (CALL or LOAD adcon group), and the original SVC is "rearmed" such that subsequent execution thereof will cause reloading of the deleted module.

Module Removal

At this point, storage is released for all nonrejected control sections of all modules to be deleted. The DEFs in each control section are removed from the TDY DEF chains, the Q-REFs in each control section are removed from the chain of Q-REFs, and the PMD itself is deleted from the TDY. This process is repeated for each module to be deleted. Unloading is complete when the last module on the deletion list has been removed from the task.

If the low-order bit of the C3 option byte is set, only the primary deletion candidate is entered on the candidate list; the tracing of implicit and explicit links is eliminated.

Unloading Example

Figure 4 shows the allocation for six modules. Module A has explicit links to B and E. Module B has explicit links to C and F. Note that module C implicitly links to D, which implicitly links to E, which implicitly links to F. If the statement DELETE B is executed from within module A, the unloading action is shown in Figure 4.

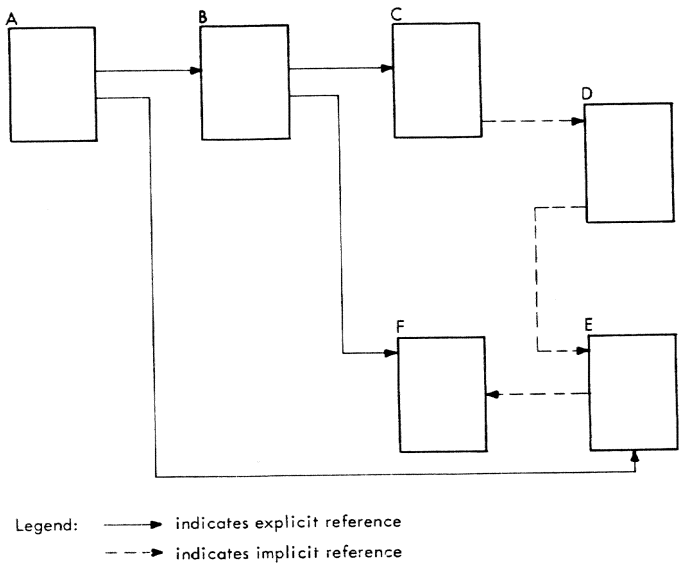


Figure 4. Unloading Example - Before

B is placed on the candidate list. B's references are traced; this results in C and F being added to the candidate list. C's references are traced; this results in D being added to the list. F has no references, so it causes no new secondary candidates to be added. Now D's references are traced, resulting in E being added to the list. E references only F, which is already on the list.

Now all modules are checked for outstanding references. B has one outstanding reference (from A); but since B is the primary deletion candidate, this explicit linkage in A is rearmed in such a way that B remains in the list. Modules C and D have no outstanding references, so they also remain. Note, however, that E has an explicit link from A that is outstanding. Thus, E is removed from the list, reestablishing the implicit link between E and F. F is examined, and it is discovered that F has an outstanding implicit reference (just reestablished from E). Thus, F is removed from the list.

At this point, modules B, C, and D are deletion candidates, and none has any outstanding references. Unloading proceeds, then, with the removing of modules B, C, and D from the task. This results in the allocation diagrammed in Figure 5.

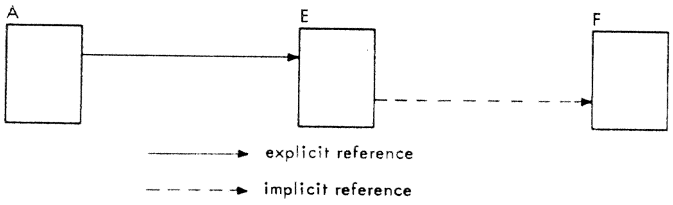


Figure 5. Unloading Example -- After

DYNAMIC LOADER CONSTRUCTION

Assembly Modules

The routines of the dynamic loader are contained in four assembly modules:

1. The loader module is composed of EXPLICIT LINKING and PAGE RELOCATION. This is the largest of the four assembly modules and bears the module name CZCDL.

2. The unloader, EXPLICIT UNLINKING, is contained in the second assembly module, CZCDU.

3. The third assembly module, CZCCD, consists of the LOADER LOGOFF, LOADER RELEASE and LOADER CLEANUP routines.

4. The last assembly module, CZCDH, consists of the LIBE MAINT service routine.

Routine Labels and Loader Entry Points

The dynamic loader is divided into small functional subroutines that bear mnemonic titles as well as coded labels. There are two types of coded labels: those that are solely internal and those that are made external symbol definitions as well by means of ENTRY statements. The internal labels all begin with the letters CGCC or CGCD. The external labels are all coded as complex definitions; the first five characters are the same as the assembly module name. For example, PAGE RELOCATION is contained in the loader module CZCDL and bears the coded label, CZCDL4.

The labels are placed in the assembly code so as to name the first instruction of the subroutine. The coded labels used to describe the loader routines in this document will coincide exactly with the coded labels in the assembly modules. Thus, the routine mnemonically titled ALLOCATE MODULE bears the label CGCCA in both this manual and in the code, while DELETE MODULE is identified (externally as well as internally) by the label CZCDU2.

The entry points of the four assembly modules of the dynamic loader are labeled as follows:

EXPLICIT LINKING	CZCDL1
EXPLICIT UNLINKING	CZCDU1
LOADER LOGOFF	CZCCD1
LIBE MAINT	CZCDH1

In addition to the four main entry points, certain other subroutines of the loader have been coded with external labels. These routines may be entered by another privileged routine using standard type-I linkage. One of the routines, LIBE SEARCH, has a macro instruction associated with it, LIBESRCH, which will expand into either type-I or type-II linkage, depending on the DCLASS of the assembly module in which the macro instruction is contained. The loader routines HASH SEARCH, LIBE SEARCH, MAP SEARCH and PAGE RELOCATION are used by other system programs and are, therefore, coded with external labels. The loader routines Q-CHAIN and RESOLVE Q-REF were put in a separate control section to reduce paging. The loader routine SET SEARCH FLAGS was made external so that it might be entered from the unloader assembly module; similarly, the unloader routine DELETE MODULE was made external so that it might be entered from the loader assembly

module. The loader logoff routines LOADER RELEASE and LOADER CLEANUP have external labels since they are called by other system programs.

Each of the four assembly modules is coded as a set of reenterable virtual storage subroutines. The executable instructions and nonvariable data are placed in control sections with the public, privileged, and system attributes. The variable data and save areas are contained in prototype control sections that have privileged and system attributes. The control sections are named using the five characters of the module name with a P or C suffixed to indicate PSECT and CSECT, respectively.

The following tables summarize the construction and content of the dynamic loader assembly modules. The names of the main entry points are underlined.

Loader Module

MODULE NAME:	CZCDL	
CSECT:	CZCDLB	(PRVLGD,SYSTEM, PUBLIC)
ENTRY POINTS:	CZCDL7	(Q-CHAIN)
OTHER ROUTINES:	CGCRQ	(RESOLVE Q-REF)
CSECT:	CZCDLC	(PRVLGD,SYSTEM, PUBLIC)
ENTRY POINTS:	CZCDL1	(EXPLICIT LINK)
	CZCDL2	(HASH SEARCH)
	CZCDL3	(LIBE SEARCH)
	CZCDL4	(PAGE RELOCATION)
	CZCDL5	(MAP SEARCH)
	CZCDL6	(SET SEARCH FLAGS)
OTHER ROUTINES:	CGCCA	(ALLOCATE MODULE)
	CGCCB	(SELECT HASH)
	CGCCC	(SRCHPACK)
	CGCCE	(RESOLVE SYMBOL)
	CGCCH	(LOAD PMD)
	CGCCJ	(FIX PMD)
	CGCCK	(ATTACH TEXT)
	CGCCL	(FIX)
	CGCCN	(ADD PMD)
	CGCCP	(REJECT DIAG)
	CGCCR	(BISEARCH)
	CGCCT	(PCSA)
	CGCCU	(CHECK DEF LEGAL)
	CGCCV	(LINK DEFS)
	CGCCW	(GET STORAGE)
	CGCCY	(DEFINE REF)
	CGCDG	(ADD MUTE)
	CGCDPR	(LOADER PROMPT)
	CGCSP	(SETPAGE)
PSECT:	CZCDLP	(PRVLGD,SYSTEM)

Unloader Module

MODULE NAME:	CZCDU	
CSECT:	CZCDUC	(PRVLGD, SYSTEM, PUBLIC)
ENTRY POINTS:	CZCDU1	(EXPLICIT UNLINKING)
	CZCDU2	(DELETE MODULE)
OTHER ROUTINES:	CGCCO	(DROP PMD)
	CGCDA	(MODIFY MUT COUNTS)
	CGCDB	(DELETE CALLER MUTES)
	CGCDC	(DELETE SELECTED MUTES)
	CGCDD	(MODUFY USE COUNTS)
	CGCDE	(TEST USER COUNTS)
PSECT:	CZCDUP	(PRVLGD, SYSTEM)

LOADER LOGOFF Module

MODULE NAME:	CZCCD	
CSECT:	CZCCDC	(PRVLGD, SYSTEM, PUBLIC)
ENTRY POINTS:	CZCCD1	(LOADER LOGOFF)
	CZCCD2	(LOADER RELEASE)
	CZCCD4	(LOADER CLEANUP)
PSECT:	CZCCDP	(PRVLGD, SYSTEM)

LIBE MAINT Module

MODULE NAME:	CZCDH	
CSECT:	CZCDHC	(PRVLGD, SYSTEM, PUBLIC)
ENTRY POINTS:	CZCDH1	(LIBE MAINT)
PSECT:	CZCDHP	(PRVLGD, SYSTEM)

Dynamic Loader Routine Linkages

The following sections discuss the basic functions of the dynamic loader: EXPLICIT LINKING, EXPLICIT UNLINKING, LOADER LOGOFF, LIBE MAINT, PAGE RELOCATION, LOADER RELEASE and LOADER CLEANUP. Each of the first three of these functions consists of a main routine and several subordinate loader routines called to support the main routine. The main routine is discussed first, followed by its subordinate routines, in the order in which they are called by the main routine. LIBE MAINT has no subordinate loader routines. PAGE RELOCATION, LOADER RELEASE and LOADER CLEANUP, although they are entry points within main modules, are not subordinate to the main routines, but are called by other system programs to perform special functions. These routines call other loader routines during their execution, and are described in the same manner as the main routines.

Figure 6 is designed to show at a glance the calling and called relationship among the dynamic loader routines. The top row of the chart shows the main entry points of the four dynamic loader modules along with the special purpose PAGE RELOCATION, LOADER RELEASE, and LOADER CLEANUP routines.

All the routines in rectangular blocks in Figure 6 are routines with external entry points, which may be entered by type-I linkage from other privileged routines. The other routines, in square blocks, are internal to their assembly modules and are

entered by restricted linkage. The INVOKE macro instruction is used for routine entrance; general registers 2 through 8 are generally considered volatile and not saved by the called routine.

Note in Figure 6 that all but two connective links are shown as solid lines in the downward direction, indicating the unilateral nature of the linkage. One exception is the upward dashed arrow from DEFINE REF to RESOLVE SYMBOL which is shown to point out the bilateral nature of this linkage. In fact, the four routines RESOLVE SYMBOL, FIX PMD, FIX, and DEFINE REF form a unique processing chain in which it is possible for DEFINE REF to enter RESOLVE SYMBOL recursively under conditions described in Section 2. Another exception is the horizontal dashed arrow from LOADER RELEASE to EXPLICIT UNLINKING, which indicates that EXPLICIT UNLINKING, one of the four loader modules, can be entered from LOADER RELEASE, a routine within another module.

The charts on the following pages supplement Figure 6 by showing, in addition to the calling relationship, the conditions prerequisite to the call. The charts are divided into levels which describe the relationship between the routines. The four main loader routines and the special purpose routines are at level 1, subordinate routines are shown at level 2 and below. This series of charts also shows the loader's interfaces with the privileged system service routines that support the loader; for example, GETMAIN, FREEMAIN, SETL, FIND, etc. Asterisks differentiate these routines from routines local to the four loader assembly modules.

Given a set of conditions, the reader can trace through the routine linkages for any processing sequence. Example:

EXPLICIT LINK to RESOLVE SYMBOL at level 1, RESOLVE SYMBOL to FIX PMD at level 2, FIX PMD to LINK DEFs at level 3, LINK DEFs to HASH SEARCH at level 4.

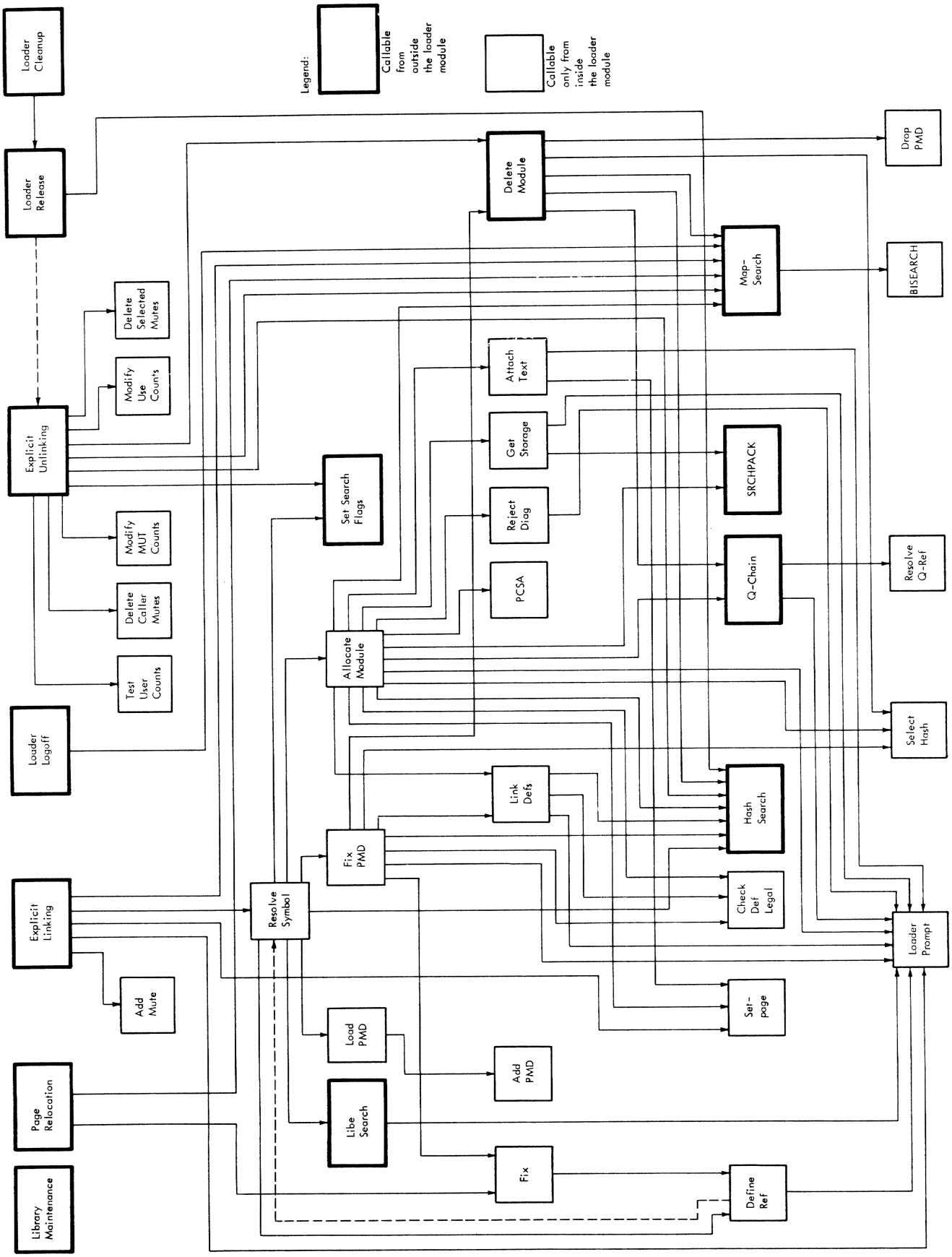


Figure 6. Dynamic Loader Routine Linkages

Routine: EXPLICIT LINK -- Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
EXPLICIT LINK	Find the definition of a called symbol and allocate storage for its containing module.	MAPSEARCH	Always called.
		RESOLVE SYMBOL	Always called.
		ADD MUTE	Called unless symbol is not found.
		LOADER PROMPT	Diagnostic when symbol not found.
		SETPAGE	SETXP request pending.

Routine: EXPLICIT LINK -- Level: 2			
Routine	Purpose	Called Routines	Calling Conditions
MAPSEARCH	Find, insert, or delete an entry in the memory MAP table.	BISEARCH	Always called.
		ABEND*	Called if MAP is full and insert requested.
RESOLVE SYMBOL	Find a DEF entry in the TDY or external library to match input argument name.	GETMAIN*	Called when next level will not fit in recursive storage page.
		SET SEARCH FLAGS	Always called.
		HASH SEARCH	Always called.
		LIBE SEARCH	Called if hash table search fails.
		LOAD PMD	Called if library search succeeded.
		ALLOCATE MODULE	Called if library search succeeded.
		FIX PMD	Called if library search succeeded.
		DEFINE REF	Called if library search succeeded and module not deleted in FIX PMD and there are yet undefined REFs.
		STOW*	Called if library search succeeded.
ADD MUTE	Construct a Module Usage Table Entry (MUTE) and disarm the calling SVC.	GETMAIN*	Called when required to expand MUT table.
LOADER PROMPT	Central routine for output of all printed matter to SYSOUT.	PRMPT*	Always called.
*Privileged system service routine external to the loader.			

Routine: EXPLICIT LINK -- Level: 3			
Routine	Purpose	Called Routines	Calling Conditions
BISEARCH	Find largest virtual storage address in MAP table ≤ input argument address.	None.	
SET SEARCH FLAGS	Determine which hash table to search for a given symbol, and if HASH SEARCH fails, which library to search.	None.	
HASH SEARCH	Find, insert, or delete a symbol in a hash chain.	None.	
LIBE SEARCH	Search a library for a module that defines a given symbol.	FIND*	Always called.
		LOADER PROMPT	Called on error return from FIND.
LOAD PMD	Transfer the PMD of a given module from a library to the TDY.	ADD PMD	Always called.
		SETL*	Always called.
		GET (LOCATE MODE)*	Called for each page of PMD.
		ABEND*	Called if PMD is invalid.
ALLOCATE MODULE	Allocate storage for each Control Section within a single module, also compute and link its absolute and relocatable DEFs.	PCSA	Called once for each control section.
		CHECK DEF LEGAL	Called for each control section name.
		SELECT HASH	If control section name is legal.
		HASH SEARCH	If control section name is legal.
		REJECT DIAG	If control section name is not unique.
		GET STORAGE	Called when all CSDs of like attributes are processed; allocates storage for control sections with same attributes.
	for fixed-length control sections	SELECT HASH	For control sections with same attributes.
		LINK DEFS	For control sections with same attributes.
		ATTACH TEXT	For control sections with same attributes, not public, or public but storage not assigned by a CONNECT.
		SRCHPACK	If CSECT packing is specified and required storage is less than a page.

Routine: EXPLICIT LINK -- Level: 3			
Routine	Purpose	Called Routines	Calling Conditions
ALLOCATE MODULE (Cont.)	for variable-length control sections	GET STORAGE	Control section of variable length.
		SELECT HASH	Control section of variable length.
		LINK DEFS	Control section of variable length.
		ATTACH TEXT	Control section of variable length, that is not public, or public but storage not assigned by a CONNECT.
	for both fixed- and variable-length control sections	LOADER PROMPT	For various diagnostics.
		MAPSEARCH	For control sections of nonzero text lengths.
		Q-CHAIN	Chain Q-REFs after assigning their values.
		SETPAGE	RESTBL of shared library is locked.
		GETMAIN*	Called to get scratch page(s) when CSECT packing is requested.
FIX PMD	Process all complex DEFs for a module, including the module named DEF.	SELECT HASH	Called unless all control sections are rejected.
		CHECK DEF LEGAL	Called unless all control sections are rejected.
		HASH SEARCH	Called unless all control sections are rejected.
		DELETE MODULE	Called when all control sections are rejected.
		LINK DEFS	Called unless all control sections are rejected.
		FIX	Called unless all control sections are rejected, if there are any complex DEFs.
		LOADER PROMPT	Diagnostic when module name rejected.
DEFINE REF	Locate a DEF entry whose name matches the input REF name.	RESOLVE SYMBOL**	Always called.
		LOADER PROMPT	Diagnostic when module name undefined or defined by complex DEF.
*Privileged system service routine external to the loader. **Recursive. See discussion of RESOLVE SYMBOL.			

Routine: EXPLICIT LINK -- Level: 4			
Routine	Purpose	Called Routines	Calling Conditions
LOADER PROMPT	Central routine for output of all printed matter to SYSOUT.	PRMPT*	Always called.
ADD PMD	Allocate space in the TDY for a new PMD.	GETMAIN*	Called when old PMD group will not accommodate new PMD to expand TDY.
PCSA	Adjust attributes of a control section according to user authority.	None.	
CHECK DEF LEGAL	Verify acceptability of an external symbol name.	None.	
SELECT HASH	Determine in which hash table a given symbol is to be posted, or in which it may be found.	None.	
HASH SEARCH	Look up, post, or delete a symbol in a hash chain.	None.	
REJECT DIAG	Examine the attributes of rejected control sections (for diagnostic purposes) and check for load error conditions.	LOADER PROMPT	Called for various diagnostics.
		ABEND*	Called when privileged control section is rejected by nonprivileged.
GET STORAGE	Request private or public storage, based on control section attributes and total pages.	GETMAIN*	Called for each private control section group.
		SRCHSDST (CZCQE)*	Called for each public control section group.
		CONNECT (CZCG7)*	Called for each public control section group when SRCHSDST returns "found."
		GETSMAIN (CZCG6)*	Called for each public control section group when SRCHSDST returns "not found."
		SRCHPACK	If CSECT packing is specified.
		LOADER PROMPT	Diagnostics for unnamed public control sections.
LINK DEFS	Post legal DEF names in a hash chain, and compute values for relocatable DEFs.	CHECK DEF LEGAL	Always called.
		HASH SEARCH	Always called.
		LOADER PROMPT	For various diagnostics.

Routine: EXPLICIT LINK -- Level: 4			
Routine	Purpose	Called Routines	Calling Conditions
Q-CHAIN	Assign values for Q-REFs and post REFs in a hash chain.	RESOLVE Q-REF	Called when no duplicate DXD name found.
		LOADER PROMPT	Diagnostic when DXDs have same name but conflicting length or alignment.
ATTACH TEXT	Set up page table entries for text pages of a control section.	LOADER PROMPT	Diagnostic if public CSECT not relocated.
		SETPAGE	Requests for external page table entries are pending.
DELETE MODULE	Delete a specified module and table entries which describe it.	SELECT HASH	Always called.
		HASH SEARCH	Always called.
		DROP PMD	Called unless module name was not found.
FIX	Process the RLDs for external REFs, internal REFs, or complex DEFs for a single page of text or PMD.	DEFINE REF	Called unless REF is already defined.
*Privileged system service routine external to the loader.			

Routine: EXPLICIT LINK -- Level: 5			
Routine	Purpose	Called Routines	Calling Conditions
CHECK DEF LEGAL	Verify acceptability of an external symbol name.	None.	
HASH SEARCH	Look up, post, or delete a symbol in a hash chain.	None.	
MAPSEARCH	Find, insert, or delete an entry in the memory MAP table.	BISEARCH	Always called.
		ABEND*	Called when MAP is full.
SELECT HASH	Determine in which hash table a given symbol is to be posted, or in which it may be found.	None.	
HASH SEARCH	Look up, post, or delete a symbol in a hash chain.	None.	
DROP PMD	Release a PMD from a PMD group.	FREEMAIN*	Called when PMD group collapses.
DEFINE REF	Locate a DEF entry whose name matches the input REF name.	RESOLVE SYMBOL**	Always called.
		LOADER PROMPT	For various diagnostics.
SRCHPACK	Search a vacant space table or create a host or symbiont entry.	GETMAIN*	Called when space not available for entry.
RESOLVE Q-REF	Assign value for a Q-REF (that is, assign offset value for the DXD).	GETMAIN*	Called to begin or expand Pseudo Vector Available Offset Table (PVAOT)
		FREEMAIN*	Called after an entire PVAOT is deleted.
LOADER PROMPT	Central routine for output of all printed matter onto SYSOUT data set.	PRMPT*	Always called.
SETPAGE	Accept and stack requests to build external page table entries, issue any pending SETXP requests, or unlock the RESTBL of a shared library.	GETNUMBR*	Incorrect member header in RESTBL.
		ABEND*	Invalid return code from GETNUMBR, RVN too large, or RPN too large.
		INTLK*	SETPAGE has not been called earlier, and library is shared.
		RLINTLK*	"Write" interlock on RESTBL header.
		SETXP*	Request for external page table entry is pending.
*Privileged system service routine external to the loader. **Recursive. See discussion of RESOLVE SYMBOL.			

Routine: EXPLICIT LINK -- Level: 6			
Routine	Purpose	Called Routines	Calling Conditions
BISEARCH	Find largest virtual memory address in MAP table ≤ input argument address.	None.	

Routine: PAGE RELOCATION -- Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
PAGE RELOCATION	Compute the correct value of adcons on the referenced page.	MAPSEARCH	Always called.
		FIX	Always called.
		FREEMAIN*	Called to release scratch pages.
*Privileged system service routine external to the loader.			

Routine: PAGE RELOCATION -- Level: 2			
Routine	Purpose	Called Routines	Calling Conditions
MAPSEARCH	Find an entry in the memory MAP table.	BISEARCH	Always called.
FIX	Process RLDs for external REFs, internal REFs, or complex DEFs for a single page of text or PMD.	None. (FIX will never call DEFINE REF when called by PAGE RELOCATION.)	

Routine: EXPLICIT UNLINKAGE -- Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
EXPLICIT UNLINKAGE	Remove a module from allocation.	MAPSEARCH	Always called.
		SET SEARCH FLAGS	Always called.
		HASH SEARCH	Always called.
		PRMPT*	If the symbol is not found. At this point exit is taken; otherwise, the following:
		DELETE CALLER MUTES	Always called.
		MODIFY MUT COUNTS	Always called.
		MODIFY USE COUNTS	Always called.
		TEST USER COUNTS	Always called.
		DELETE SELECTED MUTES	Called unless all candidates are disqualified.
		DELETE MODULE	Called unless all candidates are disqualified.
		PCS UNLOAD*	Called if PCS "AT" statements were inserted.
*Privileged system service routine external to the loader.			

Routine: EXPLICIT UNLINKAGE -- Level: 2			
Routine	Purpose	Called Routines	Calling Conditions
MAPSEARCH	Find an entry in the memory MAP table.	BISEARCH	Always called.
SET SEARCH FLAGS	Determine which hash table to search for a given symbol.	None.	
HASH SEARCH	Find a symbol in a hash chain.	None.	
DELETE CALLER MUTES	Delete all MUTEs for explicit CALLs to a specified module.	None.	
MODIFY MUT COUNTS	Increment or decrement the MUT count in the PMDs of all modules explicitly called by a given module.	None.	
MODIFY USE COUNTS	Increment or decrement, for every REF in a specified PMD, the use count in the CSD which contains the referenced DEF.	None.	
TEST USER COUNTS	Test a specified PMD for any explicit CALLs or implicit references.	None.	
DELETE SELECTED MUTES	Delete all MUTEs for explicit calls by a specified module.	None.	
DELETE MODULE	Delete a specified module and table entries which describe it.	MAPSEARCH	Always called for deletion of module name.
		SELECT HASH	Called for each nonrejected CSD.
		HASH SEARCH	Called for each nonrejected CSD.
		Q-CHAIN	Called for deletion of Q-REFs from selected hash chains.
		FREEMAIN*	Called for each nonrejected nonpublic CSD.
		SRCHSDST (CZCQE)*	Called for each public CSD group.
		FREEMAIN*	Called for each public CSD in group of like CSDs when SRCHSDST returns with user count zero.
		DROP PMD	Always called.
		DISCONNECT*	Called for each public CSD in group of like CSDs when SRCHSDST returns with user count nonzero.
*Privileged system service routine external to loader.			

Routine: EXPLICIT UNLINKAGE -- Level: 3

Routine	Purpose	Called Routines	Calling Conditions
BISEARCH	Find largest virtual storage address in MAP table ≤ input argument address.	None.	
SELECT HASH	Determine in which hash table a given symbol may be found for deletion.	None.	
HASH SEARCH	Find a symbol in a hash chain.	None.	
DROP PMD	Release a PMD from a PMD group.	FREEMAIN*	Called when PMD group collapses.

*Privileged system service routine external to the loader.

Routine: LIB MAINT -- Level: 1

Routine	Purpose	Called Routines	Calling Conditions
LIB MAINT	Maintain DCBs for the program libraries accessible by the loader within a single task.	GETMAIN*	Called when required to expand number of available DCBs.
		OPEN*	Called on the "add" function.
		CLOSE*	Called on the "delete" function.
		GATWR*	Diagnostic on close when unable to match JFCB pointers.
		SHARE*	Mark USERLIB as shared in catalog.

*Privileged system service routine external to the loader.

Routine: LOADER LOGOFF -- Level: 1

Routine	Purpose	Called Routines	Calling Conditions
LOADER LOGOFF	Called at task end to clean up the SDST.	SRCHSDST (CZCQE)*	Called for each CSD with "public name" bit on.
		DISCONNECT (CZCG8)*	Called for each nonrejected public CSD group when SRCHSDST returns with user count nonzero.
		FREEMAIN*	Called for each nonrejected public CSD group when SRCHSDST returns with zero user count.
		MAPSEARCH	Always called.

*Privileged system service routine external to the loader.

Routine: LOADER LOGOFF -- Level: 2			
Routine	Purpose	Called Routines	Calling Conditions
MAPSEARCH	Called to find virtual storage address for control section name.	BISEARCH	Always called.

Routine: LOADER RELEASE -- Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
LOADER RELEASE	CALLED BY RELEASE to unload modules before releasing DDEF; by LOADER CLEANUP to unload non-IVM modules.	EXPLICIT UNLINKAGE	Called if TDTBLK count is nonzero.
		HASH SEARCH	Called to look up module in Hash Table.
		PRMPT*	Diagnostic for each module not unloaded.
*Privileged system service routine external to the loader.			

Routine: LOADER RELEASE -- Level: 2			
Routine	Purpose	Called Routines	Calling Conditions
EXPLICIT UNLINKAGE	Called to unload non-IVM modules.	Described under EXPLICIT UNLINKAGE.	

Routine: LOADER CLEANUP -- Level: 1			
Routine	Purpose	Called Routines	Calling Conditions
LOADER CLEANUP	Called by LOGOFF to unload all modules loaded during an express batch subtask.	LOADER RELEASE	Called to unload modules.

Routine: LOADER CLEANUP -- Level: 2			
Routine	Purpose	Called Routines	Calling Conditions
LOADER RELEASE	Called to unload modules.	Described under LOADER RELEASE.	

EXPLICIT LINKING's function is to resolve an adcon group by filling in the V-value and R-value of the symbol whose alphameric name appears in the adcon group. The routine RESOLVE SYMBOL is called by EXPLICIT LINKING to define the named symbol. There are three possible outcomes to RESOLVE SYMBOL's processing:

1. The symbol is already defined in the TDY.
2. The symbol is located in an external library.
3. The symbol is undefinable for this task.

Case 1 entails no loading action, but case 2 will cause the complete loading mechanism to add the defining module to the task. The following discussion outlines the symbol resolution and module loading process which is the very heart of EXPLICIT LINKING (see Figure 7).

The first step in the symbol resolution process within RESOLVE SYMBOL is the call to SET SEARCH FLAGS. This routine sets the hash table pointer for the later call on HASH SEARCH, and sets a library index (a DCB pointer) for LIBE SEARCH in the event that HASH SEARCH cannot find the symbol defined in the TDY. Now HASH SEARCH is called to attempt to locate the argument symbol in the TDY hash table selected by SET SEARCH FLAGS. If the symbol is found, the resolution process is complete. If the symbol is not found, then LIBE SEARCH is called to locate the symbol by executing successive FINDs on each of the DCBs in the chain, beginning with the DCB selected by SET SEARCH FLAGS, until the symbol is found or the chain exhausted. If the chain is exhausted, the argument symbol is undefinable for the current task. If the symbol is located, the module loading process is put into motion.

The first step in loading is the transferring of the found module's PMD from the partitioned data set into the task dictionary (TDY). This function is performed by LOAD PMD, which further calls on ADD PMD to allocate space in the TDY.

At this point RESOLVE SYMBOL calls on ALLOCATE MODULE, which examines each control section dictionary (CSD) in the PMD. ALLOCATE MODULE collects fixed-length CSDs with the same attributes into single groups. PCSA is called to adjust the attributes in each CSD, in accordance with the rules summarized in Appendix A. Control sections not named uniquely within the selected hash chain are rejected, and REJECT DIAG is called to diagnose the conditions causing the rejection for possible anomalies warranting messages to the user.

SRCHPACK is called if private fixed-length control section packing is requested and the amount of storage required is less than a page. Otherwise GET STORAGE is called to request virtual pages to satisfy the storage requirements for either (a) groups of fixed-length control sections of like attributes, or (b) individual variable-length control sections. The legal DEFs in each nonrejected CSD are linked into the appropriate hash chains by LINK DEFs, and the page table entries for each nonrejected private control section are set up a text page at a time by calls on ATTACH TEXT. Pages from a public control section processed in the current task, but allocated to a public segment in another task, are not "attached." However, if the current task is the first to process a public control section, ALLOCATE MODULE will call ATTACH TEXT to set up the shared page table entries for the public pages that have been assigned to a public segment.

During the loading activity, RESOLVE SYMBOL will call on FIX PMD to process the complex DEFs in the new module. FIX PMD in turn will call upon FIX to execute the complex DEF RLD modifiers. FIX will process the modifiers by applying REF values to the complex DEF value words. FIX in turn will call on DEFINE REF to produce the value of any undefined REF, and DEFINE REF in its turn will call upon RESOLVE SYMBOL to obtain that definition. This defines a recursive entrance to RESOLVE SYMBOL, since it is called from within the nest of routines called by RESOLVE SYMBOL prior to the point of standard exit from the nest back to RESOLVE SYMBOL.

If the symbol to be defined on this recursive entrance is found in an external library, the loading process is begun again. This recursive process will be repeated as long as any loaded module contains a REF that is satisfied by a DEF in a module that requires loading. Only four routines are contained in the recursive chain: RESOLVE SYMBOL, FIX PMD, FIX, and DEFINE REF. Since each of these routines

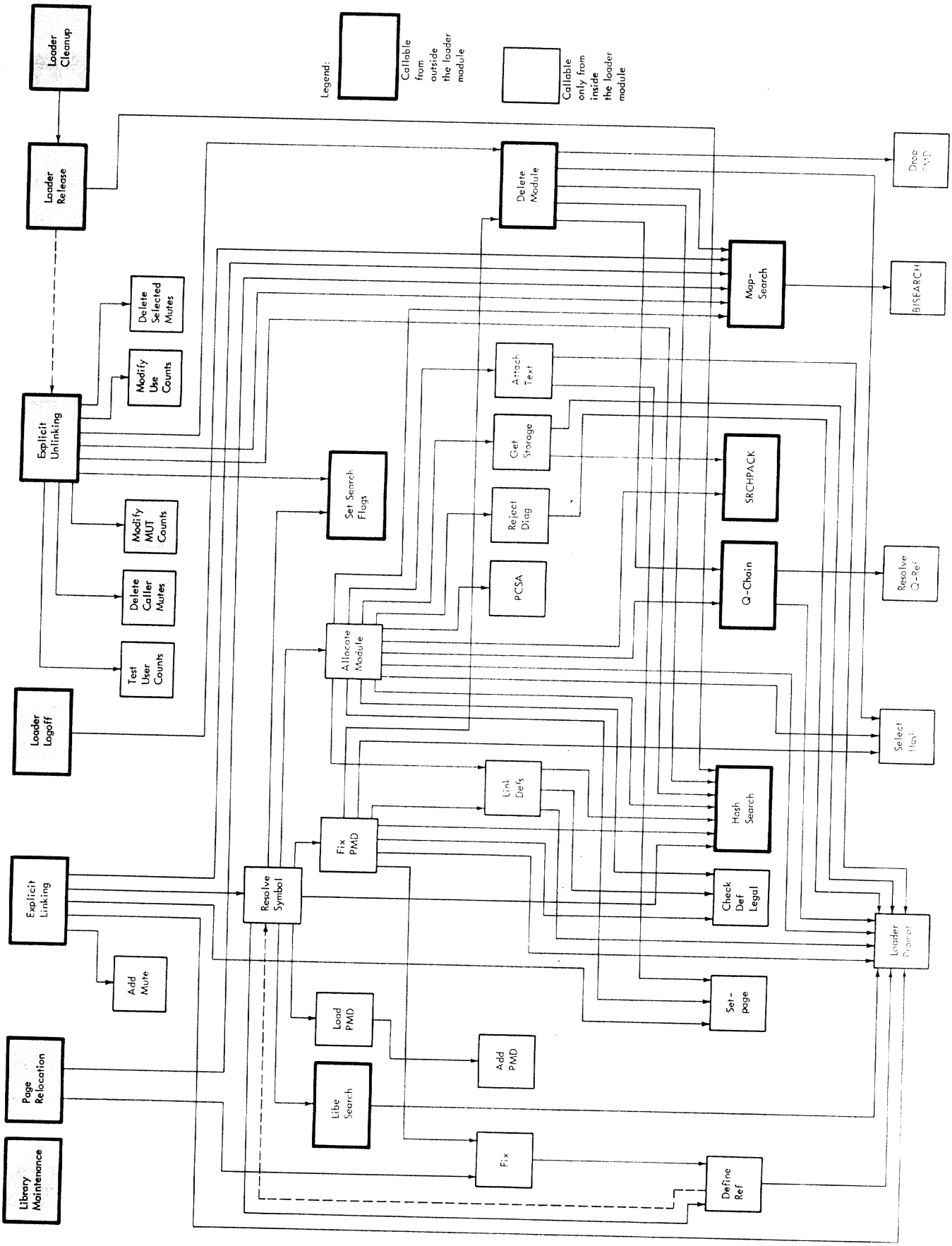


Figure 7. Explicit Linking

may be reentered to process a new module prior to completion of its processing of the "current" module, variable storage describing the processing of the current module must be maintained uniquely at each recursive level. The contents of this recursive storage and the mechanism for its allocation are described under "Resolve Symbol."

When RESOLVE SYMBOL returns to EXPLICIT LINKING via the not found exit, it has been determined that the symbol is undefinable for the task. There are three possible causes for this inability of the loader to resolve the V- and R-values of the name symbol:

1. The symbol is not defined in the TDY or in any partitioned data set that is part of the program library hierarchy available to the task at the time EXPLICIT LINKING was entered.

2. The symbol <u>is</u> defined in the TDY, but in one of the opposite hash tables -- a condition tantamount to the symbol's not being available at all. For example, the user attempts to make explicit linkage to some symbol, say CEZYK, whose module is loaded form SYSLIB, and whose defining control section is marked with the system attribute. In this case, the module will be loaded, and all the module's symbols will be posted in the appropriate system hash table according to the rules summarized in Appendix A. Symbols in the system hash tables, excepting those beginning with SYS, are not available for linkage by the user, so the explicitly names symbol is undefinable for this task.

3. The symbol is located in one of the partitioned data sets other than SYSLIB, and the defining module loaded. In the process, the control section containing the adcon argument symbol is rejected. In this case, the module loading will be completed after which RESOLVE SYMBOL will discover that the symbol is not to be found in the TDY and the not-found exit to EXPLICIT LINKING is made.

In the event of a found return from RESOLVE SYMBOL to EXPLICIT LINKING, the adcon V-value and R-value are correctly filled in, and the adcon group is availabl for linkage.

A general flow of explicit linkage is presented in Figure 8. It is divided into functional segments, rather than specific routines, to aid the reader in following the explicit linking process.

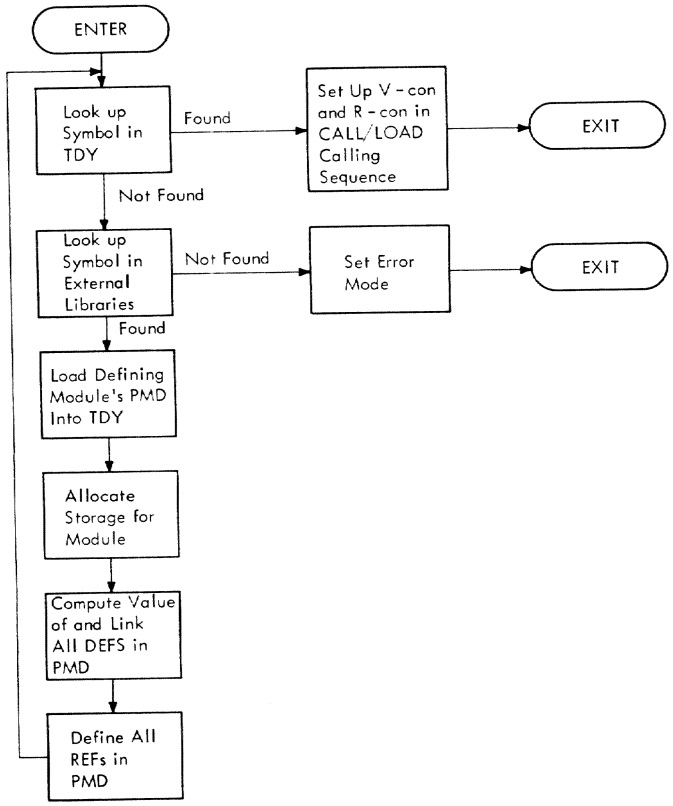


Figure 8. Functional Diagram of Explicit Linking

EXPLICIT LINK (CZCDL1)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
MAP SEARCH	Locate calling CSD.	Address of adcon group.	Pointer to CSD of control section containing adcon group.
RESOLVE SYMBOL	Provide value of symbol named in adcon group.	Name of symbol.	Pointer to defining DEF entry in TDY.
ADD MUTE	Extend BABY chain for called module; extend PAPA chain of calling module.	Calling PMD address, Called PMD address, Adcon group address.	None.
LOADER	Diagnostic on undefined argument symbol.	Pointer to parameter string.	
SETPAGE	Issue a pending SETXP request.	Pointer to parameter string; function code is a parameter.	None.

EXPLICIT LINK is called by the task monitor in response to an SVC executed as part of an explicit LOAD or CALL macro instruction. Its function is to provide the value of some symbol and, in the process, allocate storage for the containing module if the module is not already a part of the current task (see Chart AL).

Attributes: Privileged, public, system, reenterable.

Restrictions: Entrance to this routine by a type-I linkage is restricted to the task monitor.

Entries: The task monitor calls EXPLICIT LINK with GR1 pointing to a single parameter that is the virtual storage address of the explicit LOAD or CALL adcon group that caused task monitor to be entered. The form of the adcon group is:

```
            CNOP 0,4
CHD&SYSNDX  SVC  127          SVC for explicit
                                loading
            DC   H'C1C2'      Option codes
            DC   CL8'name'    Module name (or
                                alias) of
                                module to be
                                loaded
            DC   (*-12)       V-value of name
                                filled in here
                                by loader
            DS   F            R-value of name
                                filled in here
                                by loader
```

Exits:
GR15 0, normal LOAD; 8, abnormal LOAD
4, normal CALL; 12, abnormal CALL

Operation: EXPLICIT LINK first disables Load Error Switch. This switch may be enabled by a variety of routines called by EXPLICIT LINK in the loading process, such enabling being in response to some detected anomaly which is always accompanied by a diagnostic to the user. EXPLICIT LINK next disables all flags used by SETPAGE and its other callers.

EXPLICIT LINK now examines the C1 option byte, checking for the setting of the high-order bit, the "XPOS" bit, and sets the transpose flag for RESOLVE SYMBOL accordingly. This flag will be used ultimately by SET SEARCH FLAGS. If the flag is set, the loader will reverse the normal search algorithm. The normal search algorithm requires that (1) adcon groups appearing in SYSTEM control sections will be resolved from the system hash table, or that search failing, from SYSLIB, and (2) adcon groups appearing in nonsystem control sections will be resolved from the user hash table, or that search failing, from any library in the program library hierarchy beginning with the last defined JOBLIB. The presence of a transpose bit effectively complements the system attribute bit of the control section containing the adcon group to be resolved. In the transpose case (1) adcon groups appearing in nonsystem control sections will be resolved from the appropriate system hash table, or that search failing, from SYSLIB, and (2) adcon groups appearing in system control sections will be resolved from the user hash table, or that search

failing, from any library in the program library hierarchy beginning with the last-defined JOBLIB.

MAP SEARCH is called with the SVC address as an argument to locate the CSD, within the TDY, of the control section in which the adcon group appeared. This information is obtained to create the proper MUT linkage from calling to called PMD and back.

Next, an attempt is made to find the value of the symbol contained in the calling sequence. This is effected by entering RESOLVE SYMBOL with the name to be found. If the symbol is, in fact, resolvable, the found exit is taken back to EXPLICIT LINK. At this point, the module containing the symbol (the symbol could be a module name) will have been allocated virtual storage, and its PMD will have been completely processed and entered into the TDY. If RESOLVE SYMBOL must load a new module from an external library into the TDY to match the symbol, the following cascading could occur: the processing of the new PMD consists of computing the value of its external REFs, if any. Should there exist a REF whose symbol name cannot be found in the TDY, an attempt is made to locate that symbol in an external library. If such a symbol is resolved in this way, the module satisfying the REF is <u>also</u> loaded.

EXPLICIT LINK constructs a MUT entry (MUTE) for this linkage. (The MUT table is discussed in Appendix B.) The <u>calling</u> module's PAPA chain is extended to include the new MUTE while the <u>called</u> module's BABY chain is extended to point to this same MUTE. This MUTE linkage is accomplished within the subroutine, ADD MUTE, whose last task it is to disarm the calling SVC. This disarming consists in replacing a LOAD SVC with a NOP. This is done so as to prevent redundant entrance to the dynamic loader in order to load an already loaded module, thus reducing system overhead on such repeatedly executed code.

Having resolved the symbol, EXPLICIT LINK fills in the V-con portion of the calling sequence with the V-value of the satisfying DEF and fills in the R-con portion of the calling sequence with the R-value of that same DEF.

Should RESOLVE SYMBOL be unable to resolve the adcon group, it will return not found to EXPLICIT LINK. In this event, EXPLICIT LINK will insert an illegal address in both the V-value and R-value slots in the adcon group so that dynamic reference to such an address will cause an addressing error interrupt in the task. EXPLICIT LINK will issue a diagnostic in the event of an unresolved adcon group.

EXPLICIT LINK's final actions are to set the return code in GR15, and to determine if there is a SETPAGE request still pending. This return code serves to inform the task monitor as to type of adcon group as well as to error condition. A normal return code is set, if the load error switch is zero, as follows:

0 = Explicit LOAD adcon group processed properly.

4 = Explicit CALL adcon group processed properly.

If the load error switch is nonzero, the C2 option byte in the adcon group is examined. If the C2 option byte value is 1, its value is set to 7 and a normal return code is set as above.

If the C2 option byte value is zero, an abnormal return code is set as follows:

8 = Explicit LOAD adcon group processed with major error.

12 = Explicit CALL adcon group processed with major error.

An abnormal return code enables the task monitor, if in conversational mode, to cause the user to be prompted for possible corrective action. A normal return code simply causes resumption of the calling program execution. If the C2 option byte value is 1 and a load error occurs, the task monitor is not made aware of it but the calling program is (C2 option byte value set to 7) and it then has the option of initiating corrective measures.

Had there been a SETPAGE request pending, EXPLICIT LINK would call SETPAGE with a function code which tells SETPAGE to issue a SETXP for pages still in the SETXP parameter stack. Following this, EXPLICIT LINK will return to the task monitor.

Table 1 summarizes all conditions that result in load error switch setting and by what loader routine the switch is set.

EXPLICIT LINK's final action is to determine if ALLOCATE MODULE posted any CXD-REFs, that is, if any of the modules loaded as a result of the explicit load contains a CXD instruction. If a CXD-REF has been found, EXPLICIT LINK places the CXD value (the largest current offset plus the length of that offset's DXD) in all CXD-REFs.

Table 1. Load Error Summary

Error Condition	Load Error Switch Setting	Routine Detecting Error
Adcon group symbol unresolvable.	1	EXPLICIT LINK (CZCDL1)
Entry point rejected because it duplicates a previously loaded control section name.	5	LINK DEFS (CGCCV)
Public control section loaded with a text page that contains adcons.	7	ATTACH TEXT (CGCCK)
Control section rejected by previously loaded entry point not a control section name.	8	REJECT DIAG (CGCCP)
Control section rejected whose text length exceeds that of previously loaded control section of same name.	12	REJECT DIAG (CGCCP)
Undefined REF.	13	DEFINE REF (CGCCY)
REF found to be defined by a yet undefined complex DEF.	14	DEFINE REF (CGCCY)
A module is loaded during the EXPLICIT LINKING process that was noted to have been created with level 2 errors or greater.	19	ALLOCATE MODULE (CGCCA)

MAP SEARCH (CZCDL5)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
BISEARCH	Search MAP table and locate MAP entry whose VMA is the highest ≤ argument VMA.	Argument VMA.	Relative index into MAP of found entry.
ABEND	Terminate task if MAP is full and insert requested.	Address of error message.	

MAP SEARCH is called with a virtual storage address either to find, insert, or delete an entry in the memory MAP table that relates to the argument address (See Chart AZ).

Attributes: Privileged, public, system, reenterable, recursive.

Restrictions: MAP SEARCH will accept type-I linkage only from other privileged system components.

Entries: Type-I linkage to MAP SEARCH is made with GR1 pointing to a parameter which is the address of the following list:

1. A virtual storage address.
2. A function code: 0 indicating lookup, 1 indicating insert, and 2 indicating delete.
3. A pointer to a CSD to be inserted in the event of the insert function; or pointer to the found CSD in the event of a lookup function.

Exits: Normal only, no return code.

Operation: A memory MAP entry consists of two words. The first word contains the virtual storage address of the base of a control section. The second word contains a pointer to the CSD of this same control section. The MAP table is maintained in ascending order of virtual storage addresses thus facilitating a binary search for lookup purposes. A MAP entry exists in

the MAP table for each nonrejected control section in the user's task whose text length is nonzero.

MAP SEARCH calls BISEARCH with the input address. BISEARCH will return with a pointer to the MAP entry whose virtual storage address is the highest one in the table that is less than or equal to the argument address.

The MAP SEARCH find (lookup) function is complete at this point. The MAP entry found will point to the CSD of the control section containing the argument address.

For the add (insert) function, MAP SEARCH moves all the MAP entries past the one returned by BISEARCH down one entry position and physically inserts the argument address and the CSD pointer supplies.

For the delete function, MAP SEARCH erases the argument MAP entry by moving all following MAP entries one entry position up to (and including) the one to be deleted.

Both the add and delete functions adjust the current MAP entry count word in the TDY heading and set the MAP-changed-flag for BISEARCH.

On the add function, MAP SEARCH checks for a full MAP (current count equal to maximum MAP) and if full calls ABEND.

Error Checks: On lookup, MAP SEARCH does not perform validity checks to see that the found control section's text bounds can contain the argument address.

BISEARCH (CGCCR)

BISEARCH is called by MAP SEARCH to find the largest virtual storage address in the MAP table that is less than or equal to the input argument address (see Chart AE).

Attributes: Privileged, public, system, reeenterable.

Restrictions: Internal to the loader module, not available to other system components.

Entries: BISEARCH is executed in-line by MAP SEARCH. GR1 contains the argument VMA.

Routines Called: None.

Exits: In-line, no return code.

Operation: BISEARCH is designed to make a minimum of K+1 lookups on the MAP table where 2**K is less than N, the current number of MAP entries, and 2**(K-1) is greater than or equal to N. BISEARCH first checks the MAP-changed-flag set by MAP SEARCH on any previous add or delete function. When the MAP has been changed, BISEARCH recomputes K in an iterative fashion.

Initial values are set:

1. j is set to K+1. j will count the number of looks.
2. i is set to 2K*8. i is the index into the MAP table; the multiplier 8 is applied because each MAP entry is eight bytes in length.
3. d is set equal to i. d is always maintained in the main loop equal to i/2 and is the increment or decrement applied to i (hence the binary nature of the search).

The main loop is begun with d divided by 2. d is not allowed to be reduced past 8 since i must be maintained in multiples of 8 in order to index the MAP table correctly. Next, the argument address is compared against the i MAP entry.

If the argument address is less than the ith MAP entry, i is reduced by d, j is reduced by 1, and the loop reentered, which will cause an "earlier" MAP entry to be examined on the next pass.

If the argument address is greater than or equal to the ith MAP entry, j is checked for terminal value (zero or less), in which case the ith MAP entry is that MAP entry whose related control section base address is the highest VMA less than or equal to the argument VMA. If j does not test for zero or less, the search is continued. i is incremented by d and checked to ensure that it has not exceeded the maximum bounds of the MAP table. If the bounds are exceeded, i is decremented by d to bring i back within the limits of the MAP table. Note that in this latter case, the next pass will examine the same MAP entry as the previous pass, but other variables will have been altered, namely, j and d, so that future passes may examine new entries, or the search will be terminated.

A minimum of K+1 passes guarantees that the correct MAP entry is found.

On exit, GR4 will contain the address of the found MAP entry, which address is relative to the origin of the MAP table.

RESOLVE SYMBOL (CGCCE)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
SET SEARCH FLAGS	Select hash table pointer and library index	Symbol name, referencing CSD	Hash table pointer, library index
HASH SEARCH	Look up argument symbol in selected hash table	Symbol name, hash table pointer	Pointer to matching DEF entry
LIBE SEARCH	FIND symbol in external library when HASH SEARCH unable to locate in TDY	Symbol name, library index	User information describing library if found
LOAD PMD	Transfer found PMD into TDY	User information	Pointer to PMD preface in TDY
ALLOCATE MODULE	Allocate virtual storage for the loaded module's control sections	PMD address	None
FIX PMD	Process complex DEFs in loaded module	PMD address	None
DEFINE REF	Define each REF in module not defined by FIX PMD.	REF pointer, CSD pointer	None
GETMAIN	Obtain storage for recursive levels	Number of Protection class (2)	Location of assigned page
STOW	Release member interlock set by FIND	DCB address	None

RESOLVE SYMBOL is called to find a DEF entry in the TDY or external library whose name matches the input argument name (see Chart BH).

Attributes: Privileged, public, system, reenterable, recursive.

Restrictions: Internal to loader assembly module, not available to other system components.

Entries: GR1 points to a list of five parameters:

1. The address of the alphameric name of the symbol to be resolved.
2. A pointer to the CSD of the control section that contains the reference (either explicit LOAD/CALL adcon group or REF entry).
3. A transpose flag, which is merely passed on to SET SEARCH FLAGS.
4. The not found exit address.
5. Exit parameter: pointer to resolving DEF entry on found exit.

Exits: Normal, or to not found exit provided as input parameters.

Operation: RESOLVE SYMBOL is called by only two routines, EXPLICIT LINK and DEFINE REF. RESOLVE SYMBOL may itself call the routines DEFINE REF, HASH SEARCH, LIBE SEARCH LOAD PMD, ALLOCATE MODULE, FIX PMD, and SET SEARCH FLAGS. RESOLVE SYMBOL begins a circular chain of routines: RESOLVE SYMBOL could call FIX PMD, which will call FIX which could call DEFINE REF to compute a REF value.

DEFINE REF will in turn call RESOLVE SYMBOL to obtain the REF value. This entrance to RESOLVE SYMBOL constitutes the recursive call described earlier. Since at the time of recursive entrance each of the routines RESOLVE SYMBOL, FIX PMD, FIX, and DEFINE REF will not have completed processing of the current module, a mechanism is provided to preserve the variable data describing the current conditions at each recursive level.

A DSECT, CHARCS, is defined in the loader module which describes all of the variable data used by the four routines requiring recursive preservation. A block of

storage which will accommodate a large number of recursive levels, is set aside in the loader module PSECT. Each time RESOLVE SYMBOL is entered, it assumes that symbolic general register RC is pointing to the beginning of the recursive storage block currently in use. When EXPLICIT LINK makes its call on RESOLVE SYMBOL, RC is set to point to the first recursive storage block. At entrance, RESOLVE SYMBOL saves general registers 11-1 in the locations defined for the purpose in CHARCS, which is covered by RC. RESOLVE SYMBOL pushes down to the next level by adding to RC the size of the recursive storage block CHARCS. By this method, new variable data generated at the new level will be saved by the four routines in the current recursive storage block, leaving intact data stored in the previous block.

When RESOLVE SYMBOL exits, it reverses the above procedure by subtracting from RC the size of the recursive storage block thus "popping-up" to the previous level.

If the amount of space needed for the current level exceeds the space available, a GETMAIN is issued to add another page to the recursive storage block.

The following DSECT describes the contents of the recursive storage block CHARCS.

```
CHARCS   DSECT                          RECURSIVE STORAGE FOR RESOLVE
*                                       SYMBOL-FIX PMD-FIX-DEFINE REF
*                                       CHAIN RC USED TO COVER,
*                                       MAINTAINED IN RESOLVE SYMBOL
*
*        FORWARD & BACK RECURSIVE STORAGE CHAINS
*
RCSFWD   DS      F                      FORWARD CHAIN POINTER
RCSBAK   DS      F                      BACK CHAIN POINTER
*
* ****** SAVE AREA FOR RESOLVE SYMBOL (CGCCE)
RCSESV   DS      7F                     SAVE RB/RA
RCSESA   DS      1F                     SAVE FOR RA
RCSESC   EQU     RCSESV+4               SAVE FOR RC

* *****  SAVE AREA FOR FIX PMD (CGCCJ)
RCSJSV   DS      6F                     SAVE RB THRU RZ
RCSJSA   DS      1F                     SAVE FOR RA

* ******  SAVE AREA FOR FIX (CGCCL)
RCSLSV   DS      6F                     SAVE RB THRU RZ
RCSLSA   DS      1F                     SAVE FOR RA

* ******  SAVE AREA FOR DEFINE REFS (CGCCY)
RCSYSV   DS      6F                     SAVE RB THRU RZ
RCSYSA   DS      1F                     SAVE FOR RA

* ******  PARAMETER LIST FOR CALLS ON RESOLVE SYMBOL
RCSFCE   DS      0F                     NAME LIST
RCSNAM   DS      1F                     LOC(INPUT SYMBOL NAME)
RCSCSE   DS      1F                     CSD POINTER OF REFERENCING CSECT
RCSLDF   DS      1F                     LOAD FLAG - 0 = NOT EXPLICIT LOAD
*                                       1 = EXPLICIT LOAD, C1(0) = 0
RCSNFE   DS      1F                     NOT-FOUND EXIT ADDRESS
RCSSYE   DS      1F                     POINTER TO DEF ENTRY ON ''FOUND''

* ******  PARAMETER  LIST FOR CALLS ON FIX
RCSFCL   DS      0F                     NAME LIST
RCSPAG   DS      1F                     POINTER TO PAGE TO BE FIXED
RCSMCT   DS      1F                     MODIFIER COUNT FOR PAGE
RCSPFM   DS      1F                     POINTER TO FIRST MODIFIER
RCSRFT   DS      1F                     POINTER TO REF TABLE
RCSLCS   DS      1F                     POINTER TO CSD OF REF TABLE

* ******  PARAMETER LIST FOR CALLS ON DEFINE REFS
RCSFCY   DS      0F                     NAME LIST
RCSSYM   DS      1F                     POINTER TO SYMBOL TO DEFINE
RCSCDS   DS      1F                     COMPLEX DEF SWITCH - NON-ZERO
*                                       INDICATES REF IS PART OF COMPLEX
```

```
*                                          DEF
RCSCSD   DS      1F                        POINTER TO CSD CONTAINING REF

* ******          MISCELLANEOUS RECURSIVE STORAGE
RCSLBX   DS      1F                        LIBRARY INDEX - 0 = LAST OPEN
*                                          JOBLIB; N+1 = SYSLIB FOR N OPEN
*                                          JOBLIBS; N = SYSULIB.
*                                          A NEGATIVE VALUE ALSO REFERS TO
*                                          SYSLIB
RCSPMD   DS      1F                        POINTER TO PMD PREFACE (R/S)
RCSCRF   DS      1F                        REF COUNT (R/S)
RCSPCM   DS      1F                        POINTER TO CURRENT MODIFIER
*                                          (FIX)
RCSRFP   DS      1F                        REF ENTRY POINTER (R/S)
RCSPME   DS      1F                        POINTER TO 1ST BYTE PAST PMD END
RCSMNL   DS      1F                        MODULE NAME CSD LINK (FIX PMD)
RCSPMJ   DS      1F                        POINTER TO 1ST BYTE PAST PMD END
RCSFCP   DS      1F                        POINTER TO 1ST CSD (FIX PMD)
RCSHTP   DS      1F                        HASH TABLE POINTER
RCSPMP   DS      1F                        POINTER TO MDFR PRT (FIX PMD)
RCSMPE   DS      1F                        POINTER TO END OF MODIFIER
*                                          POINTERS
RCSREF   DS      1F                        REF POINTER FOR FIX
RCSFMC   DS      1F                        MODIFIER COUNT STORAGE (FIX)
RCSSNM   DS      CL8                       CURRENT MODULE NAME (DEFINE REF)
RCSBKL   EQU     *-CHARCS
RCSMAX   EQU     30
```

RESOLVE SYMBOL calls SET SEARCH FLAGS which:

1. Sets up the hash table pointer for HASH SEARCH.

2. Sets the library index for LIBE SEARCH, which determines the starting point in the chain of libraries to be searched in the event HASH SEARCH fails to find the symbol in the selected hash chain.

At this point HASH SEARCH is called to look up the symbol to be resolved. If the symbol already exists in the hash chain, the found return is made, and RESOLVE SYMBOL's job is done.

In the event that the symbol is not found in the hash chain, then RESOLVE SYMBOL must proceed to look for the symbol in the libraries according to the index set earlier. LIBE SEARCH is called, and if it returns not found, RESOLVE SYMBOL takes its not found exit; that is, the symbol is undefined.

If LIBE SEARCH is able to locate the symbol in one of the libraries, there proceeds the chain of events that effects the loading and processing of the module that defines the symbol. This takes place in four steps:

1. LOAD PMD is called to transfer the PMD of the defining module into the TDY. When LOAD PMD returns, RESOLVE SYMBOL sets the "loaded by Dynamic Loader" flag in the PMD.

2. The ALLOCATE MODULE routine is called, which effects the following:

   a. All the control sections are allocated storage (except those that are rejected for duplicate or illegal control section name).

   b. The absolute and relocatable DEFs are linked into the hash chain, and the relocatable DEFs are relocated.

   c. Page table entries are made for nonrejected control sections.

3. An R-type STOW macro instruction closes the member and releases the interlock set by FIND.

4. Following this action, RESOLVE SYMBOL enters the routine FIX PMD which:

   a. Links all complex DEFs into the hash chain and computes their value.

   b. Computes the complex DEF for the standard entry point (module name).

5. Each REF that was not defined during execution of the complex DEF modifiers in FIX PMD is computed by DEFINE REF; that is, its V-value, R-value and CSD link are filled in. This routine calls RESOLVE SYMBOL for each undefined REF (thus the need for the previously described recursive design).

If, in its processing, FIX PMD discovers that all the module's control sections were rejected, it sets a return code that forces RESOLVE SYMBOL to loop back to the call on

LIBE SEARCH to continue searching in the program library hierarchy.

Following a normal return from FIX PMD, RESOLVE SYMBOL proceeds to make a linear pass on the REF table in each CSD of the module to define those REFs not defined during FIX PMD's processing. DEFINE REF is called during this sequence, which will again recursively call RESOLVE SYMBOL.

It is possible during this module loading sequence that although the original argument symbol to RESOLVE SYMBOL was contained in the loaded module, the symbol's containing control section could have been rejected, thus rejecting the symbol definition as well. It could also be the case that the defining module was loaded but in the opposite hash table to that originally selected by SET SEARCH FLAGS. In either of these cases, the symbol is, in effect, undefined. To accommodate this phenomenon, when RESOLVE SYMBOL completes the REF processing, it executes the HASH SEARCH call once again to verify that the defining symbol was not lost in the PMD loading sequence.

In the event that the symbol is so lost, LIBE SEARCH will be called again to search the next library to attempt to find the symbol. The library index is undisturbed on exit from LIBE SEARCH to allow this sequential library search, beginning with the library following the last searched.

On its found exit, RESOLVE SYMBOL places a pointer to the first word of the defining DEF entry in the exit parameter cell.

On both the found and not found exits, RESOLVE SYMBOL pops-up its storage to maintain recursive integrity.

SET SEARCH FLAGS (CZCDL6)

SET SEARCH FLAGS is called to determine in which hash table to search for a given symbol and in which libraries to search in the event of HASH SEARCH failure (see Chart BJ).

Attributes: Privileged, public, system, reenterable.

Restrictions: Will accept type-I linkage only from other privileged system components.

Entries: On entrance to SET SEARCH FLAGS, GR1 contains the address of a parameter that points to the following list:

Inputs

1. A pointer to the argument symbol.
2. A pointer to the CSD of the control section that contains the reference symbol, which is contained in either (a) an explicit CALL/LOAD adcon group, or (b) a REF entry arising from some adcon.
3. The transpose flag, which when nonzero indicates that transposed search flags are to be set.

Outputs

1. A pointer to the hash table to be searched. This parameter will ultimately be used by HASH SEARCH.
2. A pointer to a DCB header that defines the first library in the program library hierarchy to be searched by LIBE SEARCH in the event of HASH SEARCH failure.

Routines Called: None.

Exits: Normal only, no return code.

Operation: The searching algorithm developed by SET SEARCH FLAGS is based on a set of variables. The first variable is the task authority code. If the authority code is either P or O, the algorithm is simple: search only the appropriate system hash table, and in the event libraries must be searched, search all, beginning with the last-opened job library.

If the authority code is U, the rules are more complex. The second variable is the name of the symbol to be resolved. If the symbol begins with SYS, it will appear only in the nonprivileged system hash table, and failing the system hash table search, only SYSLIB will be searched by LIBE SEARCH.

The third variable is the SYSTEM attribute bit of the calling control section; that is, the control section containing either the explicit LOAD/CALL adcon group or the REF. If the control section is a SYSTEM control section, the normal case is to search only the appropriate system hash table, or, that search failing, LIBE SEARCH will search only SYSLIB. If the calling control section does not have the SYSTEM attribute set, then HASH SEARCH will search only the user hash table, or, that search failing, LIBE SEARCH will search the entire hierarchy of open libraries beginning with the last-opened job library.

If the transpose flag is set, the effect of the SYSTEM attribute as described above is reversed.

HASH SEARCH (CZCDL2)

HASH SEARCH is called to look up, post, or delete a symbol in a hash chain (see Chart AQ).

Attributes: Privileged, public, system, reenterable.

Restrictions: Accepts type-I linkage only from other privileged system components.

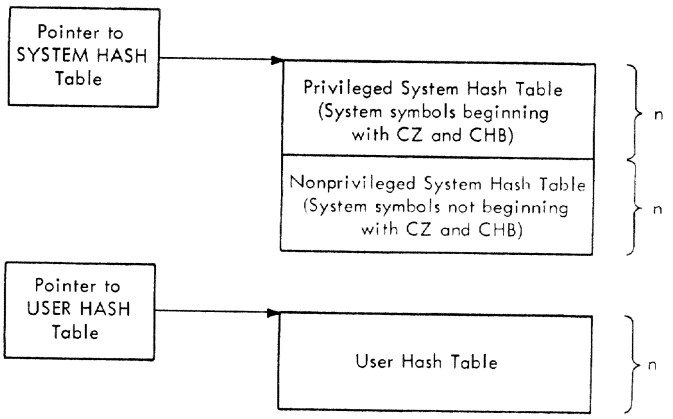
Entries: On entrance to HASH SEARCH, GR1 contains the address of a parameter which points to the following list:

1. Hash table pointer, which points to either the privileged system or user hash table.
2. The function code, which tells whether to look up(0), post(1), or delete (2) the argument symbol.
3. A pointer to the symbol name, which is either a REF entry or name part of an explicit CALL/LOAD adcon group on lookup, or a DEF entry on post or delete.
4. The module sequence number of the module containing the name in parameter 3.
5. Exit parameter. (See text.)

Routines Called: None.

Exits: Normal only, no return codes.

Operation: The collection of all nondeleted DEFs in the TDY constitutes a symbol table. To speed symbol lookup, a hashing scheme is employed as follows: There exist within the TDY three hash tables of length n, which length is set in the TDY by STARTUP as a system parameter. The three hash tables are the user hash table and the privileged and nonprivileged system hash tables. Each is a list of pointers to the heads of a possible n chains of DEFs. The member DEFs of each chain share their hash value in common. These linear hash chains are built by the HASH SEARCH "post" function.



The hash value of a given DEF is computed by performing an "exclusive OR" of the first four characters of the symbol with the last four characters. This value is then divided by n, and the remainder is the index into the hash table. This index is multiplied by 4 and added to the base address of either the privileged system or user hash table, according to the first input parameter. If a system symbol begins with CZ or CHB it is privileged and will be processed in the privileged system hash table. For nonprivileged system symbols, an additional offset equal to the size of the privileged system table is added.

The problem of relating internal REFs that reference unnamed control sections to the correct control section is solved within HASH SEARCH. Recall that unnamed CSECTs are given a name of binary zeros by the assembler. DEFs for and REFs to such unnamed CSECTs are rendered unique in HASH SEARCH by replacing the low-order half of the first word of the name of zeros with the module sequence number, a unique number assigned the module in LOAD PMD. (The second word of the zero name is reserved for the linkage editor's use to render such names uniquely relatable to their defining unnamed CSECT in the event of link editing more than one unnamed CSECT.)

The basic search function in HASH SEARCH proceeds as follows:

1. The hash value is computed.
2. The head of the hash chain in the correct hash table is obtained. (Both the head of the hash chain and all search links are the 32-bit virtual storage addresses of the first word of the next DEF in the chain.)
3. Each DEF Name in the hash chain is compared with the argument name for a match.
4. The search terminates on either a name match or at the end of chain, which is denoted by a zero search link.

On the lookup function, if no name match is found, the exit parameter is set to zero. When a name match is found on lookup, this exit parameter is set to point to the defining DEF entry.

On the post function, if no match is found, the new DEF is inserted into the top of the chain. A pointer to the last DEF in the old chain is placed in the search link of the new DEF. (The search link of the first DEF placed in the chain is zero to mark the end of the chain.) The exit pa-

rameter is set to zero to denote posting performed.

If on the post function a name match is found, the new DEF is not inserted in the chain, and the exit parameter is set to point to the duplicate DEF entry.

On the delete function, the hash chain must be relinked. Relinking is accomplished by changing the search link in the previous DEF in the chain to point to the DEF in the chain immediately following the DEF to be deleted.

LIBE SEARCH (CZCDL3)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
FIND (CZCOJ)	LOOK UP ARGUMENT SYMBOL IN opened data set.	Symbol name, DCB pointer.	User information describing found module.
LOADER PROMPT	Diagnostic on FIND error.	Pointer to parameter string.	

LIBE SEARCH is called to locate a program module in an external library that defines a certain symbol (see Chart AS).

Attributes: Privileged, public, system, reenterable.

Restrictions: LIBE SEARCH accepts type-I linkage only from other privileged system components; the LIBESRCH macro instruction is associated with it and will expand into type-II linkage for DCLASS USER.

Entries: LIBE SEARCH is entered with GR1 pointing to a parameter that contains the address of the following list:

Inputs

1. Pointer to argument symbol to be defined.

2. Library index; that is, pointer to DCB header of first DCB in the program library hierarchy to be searched. (This may also be considered an output parameter since it is modified by LIBE SEARCH during its processing.)

3. Caller-supplied location where LIBE SEARCH will place its user information on a found exit.

Output

DDNAME of library in which the argument symbol was found is placed in this location (8 bytes) by LIBE SEARCH.

Exits: GR15 = 0 normal, 4 not found.

Operation: LIBE SEARCH begins its processing by executing a FIND macro instruction, with the argument symbol to be defined, on the DCB defined by the second input parameter. Both a zero and an X'14' return code from FIND are treated by LIBE SEARCH as successful. In these cases, LIBE SEARCH will have received back from FIND 24 bytes of information describing the module as a partitioned data set member. This includes the retrieval address and length of the module's PMD, text, and ISD.

The loader will attempt to verify that the "user data" received from the FIND actually represents the user data from an actual module, and not some other kind of partitioned data set member. In the user data there are six words giving the relative page position (in the member) of the PMD, TEXT, and ISD, including their respective lengths in bytes. These are ordered as follows:

Word 1 - Relative page number of the PMD
Word 2 - PMD length
Word 3 - Relative page number of the TEXT
Word 4 - TEXT length
Word 5 - Relative page number of the ISD
Word 6 - ISD length

The following checks will be made in LIBE SEARCH after retrieving a "module."

- The user data is tested for correct length (24 bytes).
- As the PMD must always be PUT first, word 1 must equal 0.
- The PMD length (word 2) must be greater than 76.

- The relative page number of the TEXT (word 3) must equal 0 (a module without TEXT) or be equal to the integer portion of (PMD length + 4095) / 4096.

- The relative page number of the ISD (word 5) must equal 0 (no ISD) or be equal to word 3 plus the integer portion of (TEXT length + 4095) / 4096.

If the member retrieved cannot be verified as a module, the member is rejected as being invalid. Otherwise, LIBE SEARCH prefixes this information with the JFCB address and the DCB address for the library. The last word of the user information block is set to nonzero if the defining library is SYSLIB. These nine words of user information are set up in the caller's area according to the third parameter. LIBE SEARCH will also return the DDNAME of the library, so that a nonprivileged user of LIBE SEARCH may construct his own DCB for possible data set manipulation.

On an unsuccessful FIND return, LIBE SEARCH will link to the next DCB in the chain and repeat the FIND call, unless the chain has been exhausted, in which case LIBE SEARCH will make a not-found exit. Input parameter 2 is modified by LIBE SEARCH to point always to the current DCB header, so that possible successive calls by the loader routine RESOLVE SYMBOL (CGCCE) will result in continued chain search.

LIBE SEARCH sets GR15 with a return code:

0 = found
4 = not found

Notes:

1. The chain of DCBs that define the program library hierarchy is built by the routine LIBE MAINT (CZCDH). The DCBs are chained through headers prefixed to the DCBs by LIBE MAINT. These headers are four words in length.

Word 0	Link to word 0 of next DCB header
1	Link to word 0 of previous DCB header
2	Pointer to JFCB for this library
3	Not used

The end of the chain is denoted by header word 0 = zero. ISA location ISAJLC points to the first DCB header in the chain, which will be for the last-defined JOBLIB. The last DCB in the chain is for SYSLIB, and ISA location ISASLP points directly to the SYSLIB DCB header.

When LIBE SEARCH is entered, the library index will point to one of these headers or possibly be zero. Whenever the library index = 0, LIBE SEARCH takes the not found exit.

2. The format of the retrieval address in the user information is as follows:

Page Number	Zero

The 2-byte page number is relative to the beginning of the member; that is, module, of the partitioned data set. Since a PMD is normally stored as the first item of the module, its retrieval address is normally zero. The length of the PMD, text, or ISD is not necessarily an even multiple of 4096, but the first byte of PMD, text, or ISD will always fall on a new page in the data set.

3. LIBE SEARCH assumes that the user information returned by FIND in fact delimits a correctly formatted TSS/360 program module, and not some other type of partitioned data set member.

4. The format of the user information returned by LIBE SEARCH is as follows:

User Information returned by FIND

Word 0	Address of JFCB for Library in Which Name was Found
Word 1	DCB Address for Library where Name was Found
Word 2	Retrieval Address of PMD
Word 3	Length of PMD in Bytes
Word 4	Retrieval Address of Text
Word 5	Length of Text in Bytes
Word 6	Retrieval Address of ISD
Word 7	Length of ISD in Bytes
Word 8	SYSLIB Switch-Zero if Library Where Name was Found is not SYSLIB, Nonzero if it is.

LOAD PMD (CGCCH)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
ADD PMD	Find space in new PMD.	Size of new PMD.	Location of new PMD preface.
SETL	Initialize DCB for fetching first page of PMD from data set.	Pointer to DCB, relative page number of first page of PMD.	
GET (locate mode)	Transfer pages of PMD from data set to buffer page.	Pointer to DCB.	
ABEND	Incorrect PMD passed.	Address of error message.	

Error Checks: Error returns from FIND result in a diagnostic by LIBE SEARCH but are otherwise treated the same as a not-found return from FIND.

LOAD PMD is called by RESOLVE SYMBOL to transfer the PMD of a module from an external library into the TDY (see Chart AY).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader assembly module; not available to other system components.

Entries: The input parameter to LOAD PMD is in GR1, which points to the user information returned by LIBE SEARCH.

Exits: Normal only, no return code.

Operation: LOAD PMD's first function is to allocate space for the new PMD in the TDY and set up the PMD preface. This is accomplished by calling ADD PMD.

The SYSLIB flag is set in the PMD preface. This indicator is used later in checking control section attributes.

The TDTBLK field of the JFCB for this library is then incremented by 1 to indicate the loading of a module from it.

LOAD PMD now transfers the PMD into the TDY, through calls to VAM SETL and locate-mode GET. Following the GET, LOAD PMD actually block-transfers the data into the allocated TDY space. The PMD is transferred a page at a time, as required by the "undefined record format" GET function. The locate-mode GET will transfer a full page into a buffer; but if the required number of bytes is less than a page, LOAD PMD will only transfer the meaningful number of bytes from the buffer into the TDY space.

For example, a PMD 396 bytes long would be loaded by GET's fetching a full page, 396 bytes of PMD plus 3700 bytes of inapplicable information. Then LOAD PMD would move only 396 bytes into the TDY.

On exit, LOAD PMD returns a pointer to the newly loaded PMD preface.

Comments: LOAD PMD compares the PMD length in the user information with the first four bytes in the PMD returned by GET. If the values are not equal, ABEND is called.

ADD PMD (CGCCN)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
GETMAIN	Get space for new PMD in TDY.	Number of pages, protection class.	Address of page.

ADD PMD is called by LOAD PMD to allocate space in the TDY for a new PMD (see Chart AB).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GR1 contains the input parameter to ADD PMD; the length of the new PMD in bytes.

Routines Called: None.

Exits: Normal only, no return code.

Operation: PMD space is allocated within the TDY on a group basis. The pointer in the TDY to the last allocated group is fetched, and this group is now examined for available space. The end-of-group pointer in the PMD group header is fetched. Available space within a group is that space between the end of group and the end of the last (or only) page of that group. If this space can contain the new PMD, it is assigned space in that area; otherwise, GETMAIN is called to fetch storage for a new PMD group which is created with the new PMD as the only member. (Note that any PMD longer than a page will begin a new PMD group.)

Data References: TDY

The module sequence number is computed and set into the PMD preface by fetching the sequence number of the last module loaded and adding 1.

In the event a new PMD group is begun, the group header is linked into the PMD group chain in both directions: the forward link points to the new PMD group; the back link to the PMD group pointer in the TDY heading, that is, it is inserted into the head of the TDY.

Now the new PMD preface is linked within the PMD group. This linkage is a circular linkage; that is, the last PMD is linked back to the PMD group header. (This circular linkage facilitates recognition of an empty PMD group, as described under "DROP PMD.")

Finally, ADD PMD clears the remainder of the PMD preface, which effects initialization of the various flags and links.

ADD PMD returns with the virtual storage address of the new PMD preface.

ALLOCATE MODULE (CGCCA)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
PCSA	Adjust attributes of control section to be allocated.	Pointer to CSD.	
CHECK DEF LEGAL	Check legality of control section names.	Pointer to control section name.	Illegal name exit.
SELECT HASH	Choose hash table pointer for posting of control section's DEFs.	Pointer to CSD.	Hash table pointer.
HASH SEARCH	Post control section names in selected hash table.	Hash table pointer, DEF for control section name.	Duplicity indication.
LOADER PROMPT	Diagnostic on illegal control section name.	Pointer to parameter string.	
REJECT DIAG	Diagnose rejected control section conditions.	Pointer to rejected CSD, pointer to rejecting DEF.	
GET STORAGE	Obtain virtual storage for control sections.	Number of pages required, control section attributes.	Address of storage assigned, number of bytes actually assigned if variable.
LINK DEFs	Compute relocatable DEF values and post relocatable and absolute DEFs in selected hash chain.	Pointer to first DEF, DEF count, pointer to containing CSD.	Pointer to end of last DEF.
Q-CHAIN	Assign values for Q-type REFs and post REFs in selected hash chain.	Address of CSD, function code (post).	
ATTACH TEXT	Set up page table entries for control section text.	CSD pointer.	
MAP SEARCH	Insert MAP table entry for allocated control section.	VMA of control section, pointer to CSD.	
SRCHPACK	Locate virtual storage for a control section group less than a page long.	Number of bytes required.	Pointer to host entry found, VMA assigned.
	Create a host or symbiont entry for the VST.	Number of bytes in the control section group and pointer to the first control section or pointer to the host entry.	Pointer to the new host or symbiont entry created.
SETPAGE	Unlock RESTBL of the shared library being used.	Pointer to parameter string; function code is a parameter.	None.

ALLOCATE MODULE is called to allocate storage for all of the control sections within a single module and to compute and link into the hash chain all absolute and relocatable DEFs in the module (see Chart AC).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GR1 contains the input parameter to ALLOCATE MODULE, which is a pointer to the PMD preface of the module to be allocated.

Exits: Normal only, no return code.

Operation: Storage allocation is based on the following algorithm:

1. Storage is requested in a single block for all fixed-length control sections of identical attributes.

2. Storage is requested for variable-length control sections individually.

3. Storage is not allocated for rejected non-PUBLIC control sections, but space is allocated for rejected PUBLIC control sections so that the allocation in different tasks is the same for shared storage.

Upon entry, ALLOCATE MODULE disables the SETPAGE flag, specifying that SETPAGE has been called during the loading of the present module.

The CSDs are examined in order, the first unprocessed CSD defining "current attributes." All CSDs whose attributes match current attributes are located and checked for possible rejection. Rejection is caused if a control section's name is already in the hash chain, or if the name is determined illegal by CHECK DEF LEGAL. If an attribute matching control section is not rejected, its length is added to the allocation sum. When all CSDs have been examined, and each one whose attributes match is marked as processed, storage is allocated by a call to GET STORAGE.

If the packing option is set for this control section group, the loader computes the storage requirement of the group allowing for all control sections in the group to be placed on doubleword boundaries.

When a non-page-aligned control section overlaps into the next page of virtual storage, since the storage requirement is computed and alocated by subgroups, the control section overlapping the page becomes the first control section in the next subgroup.

If packing of private control sections is requested and the group length is less than one page, an attempt is made to allocate space from a partially filled page belonging to another control section of the same storage-protection class. SRCHPACK is called to search the vacant space table (VST). If an adequate vacant area is found, allocation is made at the specified VMA, and the VST is updated to reflect the assignment. If an adequate vacant area is not available, a full page is allocated by a call to GET STORAGE as mentioned above.

At this point, DEFs are computed and linked for each control section just allocated. LINKDEFs is called once for absolute DEFs and once for relocatable DEFs, with the base address allocated for the control section as an argument in each case. Absolute DEF values are, of course, unmodified, but relocatable DEFs are computed by adding to the DEF value the base address of the control section. Next, private storage page tables are set up for each control section separately by a call on the ATTACH TEXT routine. Finally, Q-REFs are assigned values and chained together by a call to the Q-CHAIN routine; CXD-REFs are also chained together. When all control sections of the current attributes group are processed as described above, another pass is made on the PMD to process control sections of different attributes.

Transfer of private packed control sections from external storage to virtual storage is accomplished by assigning scratch pages and creating entries in the VST. If packing is requested, a host or symbiont VST entry is created for each control section in the group immediately before ATTACH TEXT is called. If storage for the group is allocated from a host entry, a symbiont entry is created for each control section in the group and is linked to the host entry for the page from which the storage has been allocated. If storage has been allocated by GETMAIN, a host entry is created for the first control section in the group and symbiont entries are created for each additional packed control section in the group. These entries are created by a call to SRCHPACK. (See the SRCHPACK routine description for a further discussion of the vacant space table, VST.)

Control sections of variable length go through the GET STORAGE, LINK DEFs, and ATTACH TEXT sequence described above, but on an individual control section basis. Also, the number of bytes actually allocated is filled in the CSD heading.

Public Storage Considerations: The allocation process for public control sections is the same as for private control sections, with these exceptions:

All storage addresses are returned to ALLOCATE MODULE by GET STORAGE. Private storage addresses are always obtained by GETMAIN calls. Public storage, on the other hand, need be obtained but once for any given control section group -- or individual variable-length control section -- by a call to GETSMAIN (Get Shared Main). Subsequent tasks loading a public control section group already allocated are merely "connected" to the allocated storage by a call to the CONNECT routine. GET STORAGE sets a flag in the CSD heading of those control sections whose public storage address was returned by CONNECT linkage, rather than by GETSMAIN linkage. If a symbiont entry for this control section already exists in the SDST, (the control section has been packed on a page obtained by GETSMAIN), the CONNECT flag is set without calling the CONNECT routine and the SPT number is filled in from the host entry. This flag is necessary for ALLOCATE MODULE's processing, for two reasons:

1. Page tables need not be set up for connected public storage; consequently, on detection of this flag for public control sections, ALLOCATE MODULE bypasses the call to ATTACH TEXT.

2. When storage is obtained by GETSMAIN, GET STORAGE returns with the newly created SDST locked to other tasks. (The locked entry is denoted by the SPT number being set to all bits on.)

This entry must remain locked to all other tasks until ALLOCATE MODULE has finished all processing for the public control section group, at which point ALLOCATE MODULE unlocks the entry by filling in the SPT number returned by GET STORAGE, which obtained it from GETSMAIN.

Another consideration for public control sections is packing; if a shared public control section is already loaded, the addressing for that control section depends on whether the control sections in the group are loaded on page boundaries or are packed. V- and R-cons are calculated for either eventuality; GET STORAGE determines the amount of storage needed (and the set of V- and R-cons to be used) by checking the CSECTs-packed flag in the SDST member entry.

The only other special public storage consideration involves the "Module Public Name Switch." This flag is disabled at the beginning of ALLOCATE MODULE's processing; that is, it is disabled once for each module. It is used and enabled only by the GET STORAGE routine, and only to obey naming conventions within the SDST.

The reader is directed to the description of the GET STORAGE routine for a detailed discussion of all aspects of public storage allocation, packing of public control sections, the SDST, etc.

Before returning to its caller, ALLOCATE MODULE checks a SETPAGE flag to determine if the RESTBL of the present job library is locked because the library is shared. If this is the case, ALLOCATE MODULE calls SETPAGE, requesting it to release the interlock on the shared library's RESTBL (Relative External Storage Correspondence Table).

Error Checks: Whenever a control section is rejected, a call is made on REJECT DIAG to issue possible diagnostic messages to warn the user. REJECT DIAG discusses these cases in detail.

Comment: Whenever a public control section group is detected all of whose control sections are rejected, the public name bit is turned off in the control section of this group in which it was originally turned on. Public storage is not allocated for such a group.

PCSA (CGCCT)

PCSA is called to adjust the attributes of a control section according to user authority (see Chart BD).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module, not available to other system components.

Entries: GR1 contains a pointer to the CSD.

Routines Called: None.

Exits: Normal only, no return code.

Operation: As a module is allocated, each control section is processed by a call on PCSA. PCSA may alter the attributes of the control section according to the following rules:

1. User authority 0: Erase PUBLIC and READONLY attributes unconditionally.

2. User authority P: Erase PRIVILEGED and SYSTEM attributes from any control section that was not loaded from SYSLIB. Erase PUBLIC and READONLY attributes unconditionally.

3. User authority U: Erase PRIVILEGED and SYSTEM attributes from any control sections that were not loaded from SYSLIB, and set the SYSTEM attribute if the PRIVILEGED attribute is set in control sections loaded from SYSLIB. Erase PUBLIC attribute if module not loaded from a shared data set.

Rules 1 and 2 above make it possible for certain system programmers to load their own copies of various routines for checkout purposes. In the case of U authority, PCSA is primarily concerned with not allowing the normal user to declare PRIVILEGED and SYSTEM control sections, to eliminate the danger of resolving improperly external references from a system routine to a user routine.

Note: PCSA may reference the SYSLIB switch and the JFCB pointer in the PMD preface. This requires that the PMD link be installed in the CSD prior to invoking PCSA.

CHECK DEF LEGAL (CGCCU)

CHECK DEF LEGAL is called to verify the acceptability of an external symbol name (see Chart AF).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: On entrance to CHECK DEF LEGAL, GR1 will point to a list of three parameters:

1. Pointer to the DEF entry whose name is to be checked.
2. Pointer to the CSD containing the DEF.
3. Reject exit address.

Routines Called: None.

Exits: Normal DEF ok; to caller-supplied location if DEF is rejected.

Operation: If user authority is O, any symbol is legal. If user authority is P, symbols beginning with CZ or CHB must be defined in modules only from SYSLIB.

If user authority is U, three checks are made:

1. Symbols beginning with SYS may be defined only in control sections with the SYSTEM attribute set.
2. Symbols beginning with CZ or CHB may be defined in any control section except a SYSTEM control section that is not also marked privileged.
3. SYSTEM control sections which are also marked privileged may define only those entry points that begin with CZ or CHB.

Illegality of a DEF name is indicated to the calling routine by CHECK DEF LEGAL's taking the reject exit.

Note: CHECK DEF LEGAL may have to examine the PMD preface of the module containing the DEF; this assumes that the PMD link has been filled in the CSD heading.

SELECT HASH (CGCCB)

SELECT HASH is called to determine in which hash table a given symbol is to be posted during allocation or in which hash table a given symbol may be found for deletion purposes (see Chart BI).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GR1 contains a pointer to the CSD for whose symbols the hash table pointer is being selected.

Routines Called: None.

Exits: Normal only, no return code.

Operation: Given a pointer to a CSD, all symbols within that CSD are posted in the same hash chain. For U authority users:

1. If the PMD containing the CSD did not come from SYSLIB, the hash table pointer is set to the origin of the user hash table.
2. If the PMD containing the CSD was extracted from SYSLIB, the system attribute bit is checked. System control sections will have their symbols posted in the appropriate system hash table; non-system control sections, in the user hash table.

P and O authority users have all symbols posted into the appropriate system hash table unconditionally.

REJECT DIAG (CGCCP)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
LOADER PROMPT	Diagnostics on rejected control section conditions.	Pointer to parameter string.	
ABEND	Abnormal termination of task.	Address of diagnostic.	

REJECT DIAG is called to examine the attributes of rejected control sections for diagnostic purposes and to check for possible error conditions (see Chart BF). The purpose of this routine is to provide the user with diagnostic information so that he may identify the cause of potential task error.

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: Entrance is made with GR1 pointing to the following list:

1. A pointer to the rejected control section's CSD.
2. A pointer to the DEF entry of the name causing rejection.

Exits: Normal with no return code, or ABEND if a privileged control section is rejected by a nonprivileged control section.

Operation: REJECT DIAG's processing consists mainly of attribute comparisons. Those events causing load errors are as follows:

1. A control section is rejected by a non-control section.
2. A privileged control section is rejected by a nonprivileged control section. This could result in a privileged routine's entering nonprivileged code inadvertently. This results in an immediate ABEND.
3. A control section is rejected whose length exceeds the length of the control section causing rejection. This condition could result in possible storage protection error if the excess portion of the rejected control is subsequently referenced.

Those events that cause only a warning message to be issued are:

1. A non-read-only control section is rejected by a read-only control section. This may result in a storage protect error.
2. A COMMON control section is rejected by a non-COMMON control section or vice-versa.
3. A COMMON control section is rejected by another COMMON control section, and the length of the rejected section exceeds that of the retained section.
4. A nonprivileged control section is rejected by a privileged control section. This may result in a storage protect error or a readout error.

GET STORAGE (CGCCW)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
LOADER PROMPT	Diagnostic on unnamed public control section group.	Pointer to parameter string.	
GETMAIN	Request private virtual storage for nonpublic control sections.	Number of pages, storage class.	VMA of storage assigned.
SRCHSDST	Open public control section group member entry in SDST.	Member name.	Code indicating whether member already open and being shared by other tasks.
GETSMAIN	Request shared storage for public control sections.	Number of pages, SPT number, storage class.	SPT number, relative page number assigned.
CONNECT	Connect current task to public control section pages if public control section group in use by other tasks.	SPT number, relative page number.	Virtual storage address.
SRCHPACK	Locate virtual storage for a control section group less than a page long.	Number of bytes required.	Pointer to host entry found, VMA assigned.
	Create a host or symbiont entry for the VST.	Number of bytes in the control section group and pointer to the first control section or to the host entry.	Pointer to the new host or symbiont entry created.

Given a set of control section attributes and total pages, GET STORAGE requests private or public storage for a control section group (see Chart AP).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GET STORAGE is invoked by ALLOCATE MODULE to obtain virtual storage for either a non-variable-length control section group or for an individual variable-length control section. GR1 will contain the address of the following parameter list:

Input:

1. The attributes of the current fixed-length control section group or individual variable-length control section.
2. The total number of pages required for the control section group or variable-length control section.
3. A pointer to the first (or only) CSD of the group.
4. A pointer to the PMD preface.

Output:

1. The virtual storage address of the virtual storage block assigned.
2. The actual number of bytes assigned if it is a variable-length control section.

Exits: Normal only, no return code.

Operation: The first step in this routine is to establish the storage key for the group. Privileged control sections are assigned storage key C, which is both read and write protected against nonprivileged users. Read-only control sections are assigned storage key B, which provides

write protection. All other control sections are assigned storage key A, which allows unrestricted user reads and writes.

Next, private fixed-length control section groups are assigned storage with a single call on GETMAIN. PSECTs are assigned storage with the system packing parameter overridden to effect unconditional packing.

GET STORAGE requests virtual storage for variable-length control sections by setting the VAR flag in GR0 prior to the call on GETMAIN. GET STORAGE computes the actual number of pages assigned in the variable request, as follows:

The ISAVAR byte in the ISA is examined. If this value is nonzero, it represents the number of pages GETMAIN has assigned in addition to the requested number. If the ISAVAR byte is zero, then GETMAIN will have allocated the number of full segments required to contain the requested number of pages. GET STORAGE makes the computation, converts the total pages assigned into bytes, and sets the second output parameter accordingly.

The assignment of storage for public fixed-length control section groups and individual variable-length public control sections is a more complex process. A system table, the shared data set table (SDST), contains entries for each open shared data set. The table also contains individual member entries that describe the control section groups (or individual variable-length control sections) allocated storage by GET STORAGE.

The following rules govern the SDST member entry naming conventions for a single module:

1. The first control section group (or individual variable-length control section) is entered in the SDST with the current module name as the member name.

2. Subsequent member entries from the module bear the name of the first (or only) named control section in the group.

3. Groups of all unnamed control sections will not be entered in the SDST, and GET STORAGE will not assign public storage for such a group.

The module public name switch is disabled at the beginning of ALLOCATE MODULE's processing. GET STORAGE checks this, and finding it disabled, sets the item Public Name equal to the current module name, thus conforming to rule 1 above. Now GET STORAGE enables the module public name switch so that any subsequent assignments for control section in the current module will conform to rule 2.

The VAM routine SRCHSDST (CZCQE) is called by GET STORAGE to search for a member entry whose name matches the item Public Name. One of the parameters for SRCHSDST is the address of the JFCB describing the data set from which the current module was loaded. When SRCHSDST finds a matching member name, it checks further for a match on data set names, as follows:

Referring to Figure 9, observe that each SDST member entry contains a pointer to the SDST data set entry describing the shared partitioned data set from which the module was loaded (the module that contained the control section group represented by the member entry). SRCHSDST checks for data set match by matching the SDST data set entry name with the data set name contained in the JFCB whose pointer is passed by GET STORAGE.

If SRCHSDST finds both member and data set entry matches, it will increment the user count in the matching member entry, set a found return code for GET STORAGE, and return a pointer to the SDST member entry. The found return from SRCHSDST informs GET STORAGE that the current control section group has been allocated shared storage by another task, and that the shared page table entries have been set up. In this case, GET STORAGE extracts the shared page table number from the SDST member entry and calls CZCG7 to CONNECT the

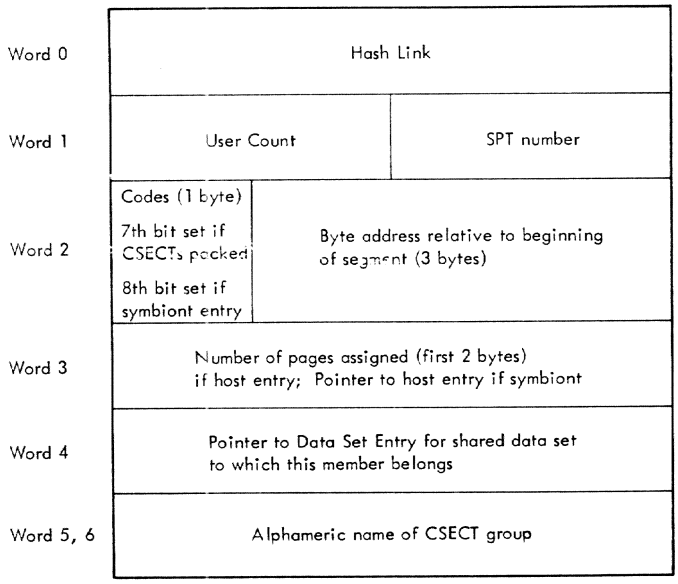


Figure 9. Sample SDST Member Entry

control section group to the shared page table. This CONNECT linkage merely converts the SPT member into a virtual storage address for this control section group for this task.

If SRCHSDST fails to find a member and data set entry match, it sets up a new entry for this control section group in the SDST and returns an indication of this fact to GET STORAGE. In this event, GET STORAGE will request public storage by calling the VMA routine GET-SHARED-MAIN (GETSMAIN). The loader always uses the previously allocated SPT number when calling GETSMAIN, such that if room exists in the previously allocated shared segment, the current request will be satisfied from that same segment. GET STORAGE fills in the new SDST entry the relative page number returned by a GETSMAIN (to be used in subsequent CONNECT calls as described above). At this point, the SDST entry is locked to other tasks by all bits having been set on by SRCHSDST in the SPT number slot in the SDST entry. Later on, the SPT number as actually returned by GETSMAIN will be inserted in the SDST entry, but only after all processing for that PUBLIC storage has been completed in ALLOCATE MODULE.

If control section packing has been specified and if SRCHSDST returns a "not found" and if the control section length is smaller than a page, an attempt is made to pack it on a partially filled page before allocating storage by GETSMAIN as described above. This is done by locking the SDST and calling SRCHPACK to thread through the vacant space table (VST), looking at the available space entry for each page. (The section on SRCHPACK describes the vacant space table.) A host/symbiont relationship exists among control section groups sharing the same page. The first group on this page is the host; the remaining groups on the page are symbionts. This relationship is established by the 'code' and 'pointer to host' entries in the SDST. For control section groups smaller than a page, when SRCHSDST has created a member entry, the vacant space table for this group's storage class is scanned for space. If vacant space is found, the group is assigned to that page, and the relevant vacant space table entry is modified to reflect the assigned space. The newly created SDST entry is flagged as a symbiont entry, and a pointer to the host SDST entry is filled in. (Host pointer is obtained from the relevant vacant space table entry.) The byte address relative to the segment is filled in, and the SPT number (obtained from host SDST entry) is saved. The user count in the host SDST is incremented and the SDST is unlocked. If vacant space is not found, allocation is made via GETSMAIN. A new vacant space entry is created if more than eight bytes of available space exist on the last page of the last control section in the group (if it is a text page). This is accomplished by a call to SRCHPACK with the function code set to create a host entry in the vacant space table. After the return from SRCHPACK, a pointer to the host SDST entry is set in the new packing table entry and the SDST is unlocked.

Error Checks: GET STORAGE will refuse to assign PUBLIC storage for an unnamed CSECT group. This condition results in:

1. A diagnostic to the user.
2. Private storage being allocated for the group.

Comments:

Note that the strict naming rules are enforced by the dynamic loader so that PUBLIC storage is allocated identically within a segment in different tasks.

The SDST is a public system table accessible by each task in the manner described above. It is the common link between tasks that effects the sharing concept.

Considering, for example, a module, M, of the following structure:

```
CSECT  A           (READ ONLY, PUBLIC)
CSECT  (unnamed)   (PUBLIC)
CSECT  B           (READ ONLY)
CSECT  C           (READ ONLY, PUBLIC)
CSECT  D           (PUBLIC)
PSECT  E           (PROTOTYPE)
CSECT  F           (PUBLIC, VARIABLE)
CSECT  G           (PUBLIC, VARIABLE)
```

The following SDST entries would appear as a result of public storage allocation for M:

An entry name "M" describing the control section group composed of A and C.

An entry named "F" describing the variable-length CSECT F.

An entry named "G" describing the variable-length CSECT G.

The module name M is used to identify the first control section group, composed of A and C. Variable-length control sections F and G are allocated storage individually, and have unique SDST entries.

The second control section group is assigned private storage, rather than public, since the first CSECT in the group is unnamed. The group is identified by the first named CSECT, D.

SRCHPACK (CGCCC)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
GETMAIN	Get space for new VST entry.	Number of pages, protection class.	Address of page.

SRCHPACK is called either to search a vacant space table (VST) for an unused storage area large enough to meet the requirements of a control section group or to create a VST entry (host or symbiont) (see Chart BL).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to the loader module; not available to other system components.

Entries: On entrance to SRCHPACK, GR1 points to the following parameter list:

Inputs

1. Number of bytes in the control section group for which storage is needed.
2. Function code (0 = search a VST, 1 = create a host entry, 2 = create a symbiont entry).
3. For function code = 1, the VMA of the first control section in the group; for function code = 2, a pointer to the host entry.

Output

Function code = 0:

1. A pointer to the host entry found.
2. The VMA assigned.

Function code = 1:

A pointer to the new host entry.

Function code = 2:

A pointer to the new symbiont entry.

Routines Called: None.

Exits:

Function code = 0:

GR15 = 0 means that space was found.

GR15 = 4 means that vacant space was not found.

Function code = 1 or 2:

Normal only, no return code.

Operation: If the function code is 0 (search a VST), the VST is searched for a text page with sufficient unused space to accomodate the group. If such a page is found, the control section group is assigned to this page and the number of available bytes in the host table entry is updated. A pointer to the host entry found is placed into word 3 of the parameter list, the virtual storage address assigned is placed into GR1, a return code of 0 is placed into GR15, and return is made. If a page with sufficient unused space is not found, a return code of 4 is placed into GR15 and return is made.

If the function code is 1 (create a host entry), an available space entry is obtained and the unused space on the last page of the last control section in the group is then computed. This is placed into the available space entry along with the page origin. This new host entry is then linked into the VST chain. A pointer to the new host entry is placed into word 3 of the input parameter list and return is made.

If the function code is 2 (create a symbiont entry), an available space entry is obtained and is linked to the host or to the last symbiont entry created for this host if any exist. A pointer to the new symbiont entry is placed into GR1 and return is made.

LINK DEFs (CGCCV)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
CHECK DEF LEGAL	Check legality of DEF name.	DEF name, CSD pointer.	
HASH SEARCH	Post legal DEFs in hash chain.	DEF entry pointer, hash table pointer.	Duplicity indication.
LOADER PROMPT	Diagnostics on DEF rejections.	Pointer to parameter string.	

LINK DEFs is called to post legal DEF names in the hash chain for a single type of DEF for a single nonrejected control section (see Chart AT). (In the case of relocatable DEFs, the value is also computed.)

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: On entrance to LINK DEFs, GR1 points to the following parameter list:

Inputs

1. DEF type code (0 is absolute, 1 is relocatable, and 2 is complex).
2. Address of the first DEF entry.
3. The number of DEFs of this type.
4. A pointer to the CSD containing the DEFs.
5. The base address assigned the control section.

Output

Pointer to the first byte past the end of the last DEF linked.

Exits: Normal only, no return code.

Operation: LINK DEFs first checks the name of the DEF in CHECK DEF LEGAL; illegal DEFs are not linked. The DEF name is looked up in the hash chain. If the name already appears and is a control section name, a diagnostic is issued if the default value REJMSG is set to N, and the load error switch is set. If the name is duplicated but is not a control section name, then a diagnostic is issued if the default value REJMSG is set to N, and the error switch is not set. Duplicate DEF names are never added to the hash chain.

DEFs surviving the above checks are now posted. First R-values are computed by adding the base address of this control section (input parameter 5) to the "R-value displacement" in the DEF entry (nonzero only if this control section's module has been link-edited and this control section combined to produce an offset).

At this point, the CSD link is set in the DEF entry. For absolute and relocatable DEFs, this is set to point to the CSD heading (input parameter 4). Complex DEFs are not so linked until later; their CSD link is set to all 1's to indicate they have been linked but not processed.

Complex and absolute DEFs are processed no further. The absolute DEF's V-value requires no change; complex DEF V-values are computed later in the FIX PMD routine. Relocatable DEFs, however, are computed by adding to the DEF value the base address of the containing control section. When all DEFs of a single type have been so processed, LINK DEFs returns with a pointer to the end of the DEF table just processed.

Error Checks: By system convention a DEF may not duplicate the name of a previously loaded control section. This condition produces a faulty load and so indicates by setting the load error switch.

Q-CHAIN (CGCQC)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
RESOLVE Q-REF	Assign DXD offset value for a Q-type REF, or designate offset as available.	A function code indicating whether the offset is to be assigned or freed, and a Q-type REF.	
LOADER PROMPT	Diagnostic when DXDs have same name but conflicting length or alignment.	Pointer to parameter string.	

Q-CHAIN processes Q-REFS. See Chart BE.

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GR1 points to the following parameter list:

1. The address of a CSD.
2. A function code (post or delete).

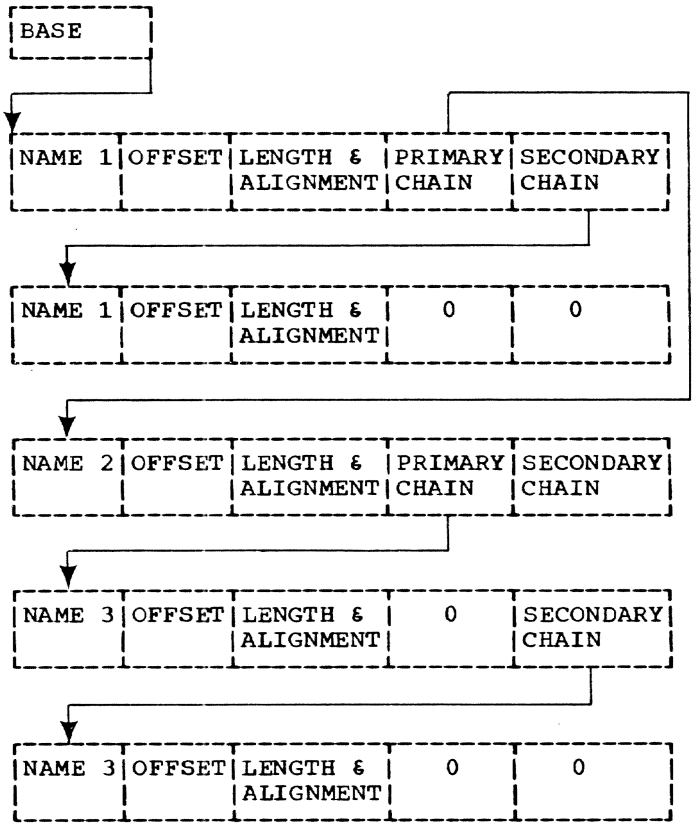
Exits: Normal only, no return code.

Operation: When Q-CHAIN is called by ALLOCATE MODULE (function code = post), the proper chain of Q-REFs is searched to determine if there is a duplicate. If a duplicate is found, the offset of the previously loaded Q-REF is assigned and the duplicate Q-REF is side chained to the previously loaded Q-REF. The length and alignment of the duplicate DXD is compared with those of the previously loaded DXD. If a conflict exists, and if the duplicate DXD requires more restrictive alignment (example: doubleword as opposed to word) or greater length than the first DXD, a diagnostic is issued:

```
CZCDL022  PROCEEDING: CONFLICTING ALIGNMENT
          OR LENGTH WITH DXD(xxxxx)
```

If no duplicate is found, Q-CHAIN calls RESOLVE Q-REF to assign an offset to the Q-REF. Once a value is assigned, Q-CHAIN chains the Q-REF into the proper primary chain.

Eleven primary hash chains are maintained for posting Q-REFs. The method of hashing is similar to that employed by the LINK DEFs routine for DEFs. When posting a Q-REF to the chains, its name is hashed and the proper chain is searched for a duplicate. If a duplicate is found in the chain, the Q-REF is not posted to the primary chain but is chained to a secondary chain. The base of this secondary chain is the sixth word of the Q-REF previously posted to the primary chain. (This form of chain is called a binary tree chain.) A typical chain might look like this:



Q-CHAIN can also be called by DELETE MODULE, to delete Q-REFs. If a Q-REF is the last one of the same name on the chain, Q-CHAIN calls RESOLVE Q-REF to reclaim the offset assigned to that Q-REF.

Once all Q-REFs in the CSD are processed, Q-CHAIN returns to the calling module.

RESOLVE Q-REF (CGCRQ)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
GETMAIN	Get page for Pseudo Vector Available Offset Table (PVAOT).	Number of pages (1); protection class (2).	Location of assigned page.
FREEMAIN	Free PVAOT space.	Address and number of pages.	

RESOLVE Q-REF assigns or frees the offset for a Q-REF. See Chart BG.

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GR1 points to the following parameter list:

1. A function code (assign or free).
2. A Q-REF.

Exits: Normal only, no return code.

Operation: RESOLVE Q-REF uses the Pseudo Vector Available Offset Table (PVAOT). PVAOT describes the storage areas available to combined dummy sections. Each entry is two words long; the first word contains the length of an area and the second word contains the offset from the beginning of the combined sections. The PVAOT entries are in order of ascending length. When searching PVAOT for an available offset, RESOLVE Q-REF finds an entry with the smallest length that will satisfy the request. It then adjusts the alignment to see if the entry will still satisfy the request. If the entry now fails, RESOLVE Q-REF serially examines the following entries until an acceptable entry is found. The needed length is subtracted from the entry's length, the offset is updated, and the table is reordered as required.

When an offset is freed, RESOLVE Q-REF updates the appropriate offset and length fields.

The space for PVAOT is acquired and initialized upon encounter of the first Q-REF. If the table becomes full, RESOLVE Q-REF expands it with the requirement that it be contiguous in virtual memory.

CGCRQ places the largest offset plus its DXD's length in the PSECT of the dynamic loader for future reference by EXPLICIT LINK (CZCDL1).

ATTACH TEXT (CGCCK)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
LOADER PROMPT	Diagnostic when public page contains adcons.	Pointer to parameter string.	
SETPAGE	(1) Process a request for an external page table entry. (2) Issue pending SETXP request when packed public CSECT text is to be moved from scratch page to unassigned VMA.	Pointer to parameter string; function code is a parameter.	None.

ATTACH TEXT is called to set up page table entries for text pages of a single control section (see Chart AD).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: Entrance to ATTACH TEXT is made with GR1 pointing to this parameter list:

1. The address of the CSD whose section text is to be attached.

2. The address of the PMD preface.

Exits: Normal only, no return code.

Operation: The virtual storage page table (VMPT) within the CSD is examined an entry at a time. This table is produced as part of the CSD by the language processor that created the module. There will be a VMPT entry for each page of virtual storage spanned by the control section, whether hard text was produced for each page or not. A VMPT entry is two bytes and may contain either:

1. X'FFFF', which indicates that no text page exists to match the virtual storage page.

2. A number, not X'FFFF', which is the number of the text page in the data set relative to the first page of the control section text in the data set member.

For example, a control section may contain two pages of code, an empty page (say, a DS 4096C, to reserve the page), and another page of data. The VMPT for such code would be:

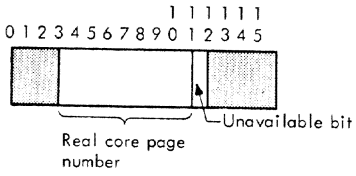
```
0000    0001
FFFF    0002
```

from which it may be seen that the first two virtual storage pages have corresponding text pages 0000 and 0001. The third page of virtual storage spanned by the control section has no corresponding text page, and the last virtual storage page corresponds to text page 0002.

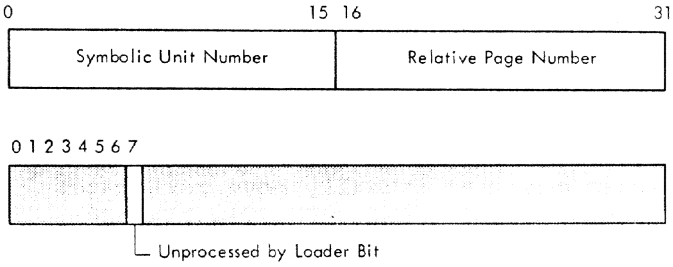
The routine SETPAGE is called to process requests for a page table entry relating the virtual storage address of a text page to the external storage address where the text page resides in the data set. The loader does not actually move text pages from their data sets into real storage. Instead, the external page table (XPT) entries are set to point to the data set text residence, while the page table entry is marked "unavailable." When the CPU makes a virtual storage address reference to a virtual page whose page table entry is flagged unavailable, an interruption occurs. The resident supervisor processes the interruption by assigning a real storage page to contain the virtual page, resetting the unavailable bit, filling in the main storage page address in the page table, and processing the corresponding XPT entry. If the XPT entry is zero, no action is required; that is, the virtual page has no corresponding external page. If the entry is not zero, it will contain the external or auxiliary storage address of the page that the resident supervisor paging mechanism will transfer into the assigned real storage page. A bit is also contained in the XPT entry that when set causes the task monitor to enter the PAGE RELOCATION entrance (CZCDL4) of the loader to process adcons on the referenced page. This bit is known as the "unprocessed by loader" bit and is set or reset by parameter when ATTACH TEXT calls SETPAGE.

For purposes of this discussion the following fields of the page tables and XPT are discussed:

1. The page table entries are one halfword. Bits 4-11 contain the main storage page address, while bit 12 is the "unavailable" bit:



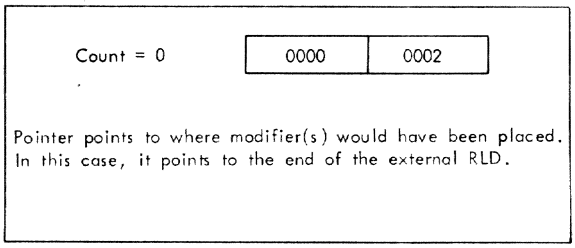
2. The XPT entry is two words in length. The first word is the external storage address expressed as symbolic unit and relative page number on the unit. The second word is a series of flag bits including the unprocessed by loader bit:



ATTACH TEXT examines each VMPT entry in the argument CSD. There will be a VMPT entry for each page of virtual storage to be spanned by the control section. This count is easily determined by rounding up the text length to the next integral page. The CSD field, TDYCTL, contains the number of bytes of virtual storage to be spanned by the control section. Adding 4095 to this value and dividing by 4096 (right shift of 12) produces the page count and VMPT entry count.

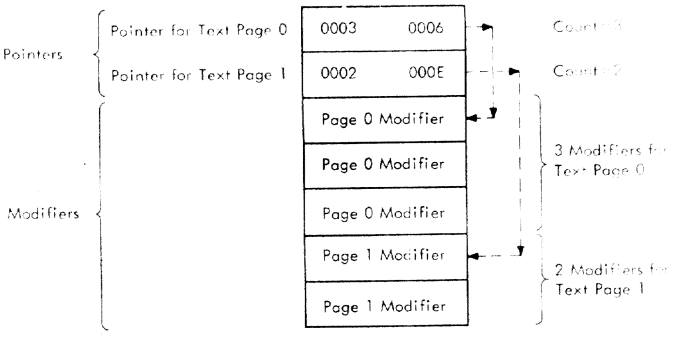
ATTACH TEXT checks the VMPT entry for all Fs, in which case the SETPAGE linkage is skipped, and the next VMPT entry is examined. When the entry is discovered to contain a relative text page number, the number is used as an index into the external and internal Relocation Dictionary (RLD) tables. There will be no RLD entry for a given virtual page if there is no corresponding text page.

The RLD tables are constructed so that the modifier pointers are in order by text page number, that is, the first pointer is for page 0 of control section text, the second for page 1 of control section text, etc. Virtual pages with no corresponding text pages have no pointers in the RLD. (Such pages cannot contain adcons, so RLD entries are unnecessary.) The RLD is kept as compact as possible by including pointers only for those text pages through the last one in the control section to contain adcons. The pointer for a text page with no adcons will point to that portion of the RLD where the modifiers would have been placed had there been any. In the case of a control section with no external adcons, for example, the external RLD would consist of a single pointer and no modifiers:



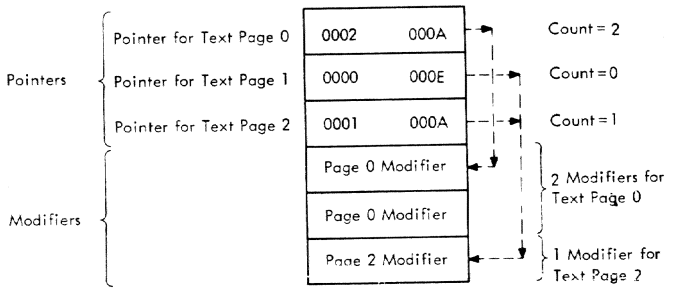
Referring back to the earlier example, assuming that only pages 0000 and 0001 contain external adcons while pages 0000 and 0002 contain internal adcons, the RLDs would look like this:

External RLD:



Note that since there are no external adcons on text page 2, there is no modifier pointer for that text page in the External RLD.

Internal RLD:



Note that text page 2 corresponds to virtual page 3, and that there is no pointer for vacuous virtual page 2.

The location of the last pointer within an RLD is determined from the fact that the last pointer and the first modifier are in adjacent words. The first pointer in each RLD points to the first modifier; that is, the location of the second half of the pointer plus the contents of the second half of the pointer locate the first modifier. ATTACH TEXT makes this computation for each RLD separately and uses this location to delimit the end of the RLD pointers.

If a text page exists, ATTACH TEXT will call SETPAGE with the unprocessed by loader parameter set, if a nonzero modifier count for the current page is discovered in either RLD. ATTACH TEXT will call SETPAGE with the unprocessed by loader bit reset in the event that in both RLDs either:

1. The current text page falls beyond the last page for which there exists a modifier pointer.
2. The current text page has an RLD modifier pointer but with a zero modifier count.

If ATTACH TEXT discovers a public page with adcons, a diagnostic is issued, the load error switch is set, and ATTACH TEXT calls SETPAGE with the unprocessed by loader bit reset.

When private control section packing has been specified and the unprocessed by loader bit is "on" at page relocation time, this informs the loader that some of the text page to be relocated may not have been loaded on the page. The last text page of any control section group (if this last page has a corresponding text page) has the unprocessed by loader bit set unconditionally since this page may be a packed one. If the control section to be attached is packed, the virtual storage address of its scratch page is obtained from the relevant symbiont entry in the vacant space table. The control section text is attached to the scratch page via a call to SETPAGE. The text is properly packed and the scratch page released at page relocation time. For the calls to SETPAGE mentioned above, ATTACH TEXT supplies a function code specifying that a request for an external page table be placed in a stack which SETPAGE maintains.

For public packing, the unprocessed by loader bit has no significance. If control sections do not begin on internal page boundaries, the adjustment must be made by ATTACH TEXT. ATTACH TEXT calls SETPAGE, giving a function code specifying that any pending requests to build external page table entries must be performed. Then the control section text is attached to a scratch page and immediately transferred to the assigned storage area in the packed page.

FIX PMD (CGCCJ)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
SELECT HASH	Select hash table pointer for posting of complex DEFs (including module name DEF).	CSD pointer.	Hash table pointer.
CHECK DEF LEGAL	Check legality module name DEF.	Module name, CSD pointer.	Illegal DEF exit.
HASH SEARCH	Post module name DEF in selected hash chain.	Module name DEF pointer, hash table pointer.	Duplicity indication.
LOADER PROMPT	Diagnostics on rejected module name.	Pointer to parameter string.	
DELETE MODULE	Remove PMD from TDY when all control sections rejected.	PMD address.	
LINK DEFS	Link complex DEFs into selected hash chain.	Pointer to first complex DEF, DEF count.	
FIX	Process complex DEF RLDs and module name RLD.	Modifier count, pointer to first modifier, PMD address.	

FIX PMD processes all the complex DEFs for a module including the module name DEF (see Chart AO).

Attributes: Privileged, public, system, reenterable, recursive.

Restrictions: Internal to loader, not available to other system components.

Entries: FIX PMD is called with GR1 pointing to the PMD preface.

Exits:

Normal - GR15 = 0: module not deleted.

Error - GR15 = 4: module deleted because all control sections rejected.

Operation: FIX PMD makes four separate passes on the CSDs of a module.

Pass 1 finds a control section to be associated with the module name (standard entry point) DEF. This involves locating the first nonrejected PSECT (or CSECT if no PSECT found) and saving its base address. During this process, if it is discovered that all control sections have been rejected, the entire module is dropped with a call to DELETE MODULE. In this case, GR15 is set to 4, and the exit is taken.

The successful locating of such a control section is followed by a legality check on the module name. If this test passes, an attempt is made to insert the name in the hash chain. If the name is either illegal or duplicately defined, a diagnostic is issued advising the user that the standard entry point is not defined for this module. If the module name is posted successfully, the CSD link in the module name DEF entry is set to X'FFFFFFFD' to flag the DEF as a module name DEF whose value is not yet computed. The CSD pointer is saved in recursive storage for later use in plugging in the module name DEF CSD link.

Pass 2 calls LINK DEFs to link the complex DEFs in each CSD into the hash chain and also to set the CSD link to X'FFFFFFFF'. DEFs not posted because of illegal or duplicate names will have CSD links = 0 on return from LINK DEFs.

Pass 3 calls FIX to process the complex DEF modifiers for each nonrejected control section.

Pass 4 examines all the complex DEFs for CSD links equal to all 'F's and changes these links to point to the defining CSD. Complex DEFs with zero CSD links will be unmodified.

This four-pass design is necessary to avoid possible looping definitions of com-

plex DEFs. Loops could occur because FIX PMD calls FIX, which, in the processes of resolving a REF for a complex DEF, might call DEFINE REF, which calls RESOLVE SYMBOL, which could again call FIX PMD. Thus, the following situation could occur:

Module A has a complex DEF, Y, referring to symbol X. Symbol X is contained in module B, which is then loaded and its complex DEFs processed; this involves a reference back to undefined symbol Y in module A. To protect against this circular definition, the device of setting the high-order five digits of the CSD link to all X'F's is employed. DEFINE REF always examines the CSD link in the defining DEF, and if it notes that the high-order five digits are all X'F's, it knows a looping possibility exists and therefore calls the symbol undefined. The low-order three digits are used to distinquish between complex DEFs and the module name DEF. Complex DEFs are identified by the low-order three digits = X'FFF'; the module name DEF is identified by X'FFD'. This distinction is made solely for diagnostic purposes, so that when any REF is defined by a DEF with such a CSD link, it will be possible to provide more detailed diagnostic information.

Note that the CSD link is not set for any complex DEF in the module until all complex DEFs have been computed by FIX. This also prevents some situations that could be resolvable but cannot be distinquished from the looping case without exceedingly costly checks.

FIX PMD's final task involves processing the complex DEF for the module name. First, the module name CSD link is checked for zero indicating rejection in pass 1; that is, the CSD link for the module name DEF is set to point to the CSD of the control section whose base defines the R-value for the DEF. In the case of module name rejection, no further processing takes place. If the module name CSD link is not zero, the R-value is installed, using the CSD pointer saved during pass 1. Once the module name R-value is set, the V-value is determined by calling FIX to compute the DEF value. The CSD link = X'FFFFFFFD' functions like the all 'F's CSD link; that is, to catch possible looping definitions in a module loading cascade started by calling FIX.

Associating the module name DEF with some control section in the module causes the module name to assume, and be governed by, the attributes of that control section. An example of such effect would be the posting of the module name DEF in the system hash table if the system attribute bit is set in the associated control section.

Comments: See "Resolve Symbol" for discussion of recursive chain of routines that includes FIX PMD.

FIX (CGCCL)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
DEFINE REF	Provide value of undefined REF.	REF pointer, CSD pointer.	

FIX is called to process the RLDs for either external REFs, internal REFs, or complex DEFs for a single page of text or PMD (see Chart AN).

Attributes: Privileged, public, system, reenterable, recursive.

Restrictions: Internal to loader module; not available to other system components.

Entries: Entrance is made to FIX with GR1 pointing to the following list:

1. A pointer to the page to be fixed
2. Count of modifiers
3. Pointer to the first modifier
4. A pointer to the base of the REF table to which the modifiers refer
5. A pointer to the CSD containing the REF entry

Exits: Normal only, no return code.

Operation: FIX is called separately for each text page to be fixed and for each type of REF, external and internal. FIX is called again to compute the complex DEF values within the PMD. This is also done by pages individually.

The first step in fixing is to fetch an RLD modifier (Figure 10), according to the RLD modifier pointers. For processing, the modifier is unpacked into its component parts. The length field, L, determines the number of bytes of text (or PMD) to be modified. A zero length field is interpreted as a four-byte modification. This length is taken to mean the number of bytes beginning with the byte pointed to by the "byte" field.

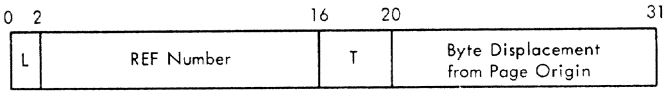


Figure 10. RLD Modifier Format

FIX now computes RN, which is the "REF number" portion of the modifier multiplied by the size in bytes of a REF entry. (The REF number is the ordinal position of the REF in the REF table. The numbering begins with zero and ascends in increments of 1.)

The modifier operation, T, determines the modification to be effected. A value of 1 means addition of the REF value to the text (or PMD) slot. A value of 2 means subtraction of the REF value from the text (or PMD) slot. A value of 3 means to substitute in the text slot the R-value of the REF.

The byte pointer, B, is the relative location within the page of the word to be modified.

Unpacking completed, FIX fetches the CSD link of the REF at the location determined by adding RN to the base of the REF table. If this CSD link is nonzero, this means the REF is already defined. The modification proceeds by performing the modifier operation as determined by T on the L right-adjusted bytes at location B plus the base of the page being fixed. The REF value to be applied is right-aligned with the text field to be modified, with high-end truncation applied for text fields less than a full word.

If the CSD link in the REF entry is zero, this indicates that the REF has not yet been satisfied. FIX proceeds then to call DEFINE REF, with the REF name as the main argument in order to set the REF's V-value, R-value, and CSD link in the REF entry. REFs will always be defined when FIX is called during page relocation.

Comments: See RESOLVE SYMBOL for discussion of recursive chain of routines that includes FIX.

DEFINE REF (CGCCY)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
RESOLVE SYMBOL	Obtain value of REF symbol.	REF name, CSD address.	Pointer to resolving DEF entry in TDY.
LOADER PROMPT	Diagnostics on REF definition anomolies.	Pointer to parameter string.	

DEFINE REF is called to locate a DEF entry whose name matches the input REF name (see Chart AG).

Attributes: Privileged, public, system, reenterable, recursive.

Restrictions: Internal to loader module; not available to other system components.

Entries: On entrance to DEFINE REF, GR1 contains the address of the following parameter list:

1. The pointer to the REF entry to be defined.
2. The complex DEF switch.
3. A pointer to the CSD containing the REF entry.

Exits: Normal only, no return code.

Operation: RESOLVE SYMBOL is called to locate a matching DEF. If the return is "not found," a diagnostic is emitted, a dummy value substituted, and the load error switch is set. A found return from RESOLVE SYMBOL is followed by a sequence of checks.

First, the CSD link of the defining DEF entry is examined for the high-order fifty digits = X'FFFFF'. If this condition occurs, a possible loop has been uncovered.

A diagnostic is issued advising that a REF has been defined by an as-yet-undefined complex DEF. Again, a dummy value is substituted, and the load error switch is set.

If the CSD link is a legitimate link, the complex DEF switch is checked. If the switch is set (indicating the processing of complex DEFs), a check is made to see if the defining DEF, although properly defined, is a complex DEF. If it is, the definition is made properly, but the user is warned. This warning is issued because following the link-editing of two modules, one of which has a complex DEF defined by a complex DEF in the other, such definition will not be possible because of the all bits protection device.

Following all diagnostic checks, the REF entry V-value, R-value, and CSD link are filled in from the defining DEF entry. Finally, the use count in the defining DEF's CSD is incremented to reflect this implicit reference.

Comments: See "RESOLVE SYMBOL" for a discussion of the recursive chain of routines that includes DEFINE REF.

ADD MUTE (CGCDG)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
GETMAIN	Get page for MUT table when current MUT full.	Number of pages (1); Protection class (2).	Location of assigned page.

A new module usage table entry (MUTE) is constructed, and the calling SVC is disarmed (see Chart AA).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: GR1 points to a parameter list with the following:

1. Location of SVC
2. Calling PMD pointer
3. Called PMD pointer

Exits: Normal only, no return code.

Operation: Space for the new MUTE is allocated in the module usage table (MUT) at the location indicated in the MUT available space pointer. If there is no available space, a new page is acquired by GETMAIN, and all of its space is linked into the available space chain. The new MUTE is constructed and linked, as follows:

1. It is linked into the PAPA chain of the PMD of the module that initiated the explicit CALL. The head of the PAPA chain in the calling PMD preface points to the new MUTE's forward PAPA link word. (See Appendix B for description of MUT linkage.)
2. It is linked into the BABY chain of the PMD that was explicitly called. The head of the BABY chain in the called PMD preface now points to the new MUTE's forward BABY link word.
3. A pointer to the called PMD is inserted in the new MUTE.
4. The virtual storage address of the SVC that initiated the CALL is inserted in the new MUTE.
5. The SVC that initiated the explicit linkage is disarmed with a NOP.
6. The MUT count in the called PMD is increased by one.

LOADER PROMPT (CGCDPR)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
PRMPT	Issue diagnostic.	Address of message ID.	

LOADER PROMPT provides a centralized routine for output of loader diagnostics through PRMPT, thus making unnecessary the repetition of the costly PRMPT linkage at each diagnostic point (see Chart AV).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to loader module; not available to other system components.

Entries: LOADER PROMPT is entered by restricted linkage.

GR1 contains the address of the parameter string.

RE contains the address of the loader PSECT, CZCDLP.

Routines Called: PRMPT, with the address of the parameter string in register 1.

Exits: Normal only, no return codes.

Operation: The address of the parameter list for the diagnostic is passed in register 1. Symbolic general register RE is used to cover the loader's PSECT into which the variable portion of the PRMPT macro instruction is expanded. Register 13 may not be covering CZCDLP, but will be covering the save area currently available when the PRMPT linkage is effected. For example, LIBE SEARCH, which calls LOADER PROMPT, operates with GR13 covering the loader's second save area; other routines that call LOADER PROMPT operate with GR13 covering the first save area, that is, the origin of CZCDLP.

SETPAGE (CGCSP)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
ABEND	(1) Invalid return code from GETNUMBR. (2) RVN found in external page entry in RESTBL too large. (3) Reference beyond data set end (RPN requested too large).	Pointer to ABEND message.	None.
GETNUMBR	Correct the "first external page entry" locator field in the member header of the RESTBL. It also converts relative page number into member to relative page number into data set.	Pointer to word with DCB address.	Relative page number of page requested into data set.
INTLK	Place "write" interlock on RESTBL header.	Pointer to a list of pointers.	None.
RLINTLK	Release "write" interlock on RESTBL header.	Pointer to a list of pointers.	None.
SETXP	Build one or more external page table entries.	Pointer to a parameter list.	None.

SETPAGE will accept requests to build external page table entries and will stack these when possible. SETPAGE will call a supervisor routine, SETXP, to have external page table entries built for contiguous virtual storage pages represented in a special parameter list. (See Chart BK.)

Attributes: Privileged, public, read-only, system.

Restrictions: Internal to dynamic loader module; not available to other system or user modules.

Entries: Entered using "INVOKE CGCSP$".

Input: A parameter list and a set of flags:

1. GR1 points to the following two-word, one-byte parameter list:

   Word 1: Pointer to a DCB (if function code 00).

   - DCBN (halfword) - specifies the relative page number into the member which is requested.
   - DCBOP (halfword) - specifies operation performed (that is, value of X'8000' is always guaranteed meaning "input"; if value of X'4000' is present (giving X'C000'), then "unprocessed-by-loader" flag setting for the requested page is wanted.
   - DCBNI (halfword) - not used by SETPAGE. It is changed by GETNUMBR when it is called.

   Word 2: VM address for page requested.

   Byte: A function code for SETPAGE:

   00 - Process a page request.

   04 - Issue a pending SETXP.

   08 - Release the write interlock on the RESTBL of the present (shared) library.

2. SETPAGE flags (DYSPFLGS):

   DYSPCALM (X'01') - SETPAGE called earlier (same module).

   DYSPSXPM (X'02') - There is a pending SETXP.

   DYSPSHRM (X'04') - The RESTBL of the present (shared) library is locked (write interlock).

Output: The desired function has been performed; one or more of the SETPAGE flags will have been set on.

Exits:

Normal - Return via RESUME macro; no return code.

Error - ABEND completion code 2, module name, and library DDNAME are supplied.

Reasons:

1. Invalid return code from CZC001 (GETNUMBR).
2. RVN (relative volume number) found in external page entry (EPE) in RESTBL was too large.
3. Reference beyond data set end - RPN (relative page number) passed in DCBN was too large for data set.

Operation: SETPAGE will examine the function code it receives and will act upon it in one of these ways:

00 - Process a page request as described later.

04 - Issue any pending SETXP request and return.

08 - Unlock the RESTBL header of a shared library and return.

Page requests (code 00) are processed as follows:

If SETPAGE has not been called yet for this member (module), SETPAGE will calculate the address in the RESTBL (of the library used) of the first external page entry for the member. Also, the limiting address for EPEs will be calculated and saved for later error checking. GETNUMBR (CZC00) will be called to adjust the member's header in the RESTBL if the member EPEs had been moved.

With the 1st member EPE address known, SETPAGE will now locate the external page entry (EPE) in the RESTBL for the page requested by SETPAGE's caller. The relative page number in the member is passed to SETPAGE. SETPAGE will use this number times the proper EPE byte size to look past the 1st member EPE for the desired page's EPE.

Having located the proper EPE, SETPAGE will place the external page number from the EPE in the next entry being built for the stack of SETXP parameters. The rela-

tive volume number (RVN) in the EPE must be used to set into the new SETXP entry the proper symbolic device address (SDA) on which to find the external page. If the RVN is the same as for the last call to SETPAGE, then the SDA last placed in the SETXP entry build area is still valid, otherwise, the RVN must be used to locate the proper SDA in the public/private volume table (PVT). This SDA found is placed in the SETXP entry being built.

SETPAGE will stack the SETXP page request entry with any preceeding requests if there are any and if the stack is not full (1021 entries). As it becomes necessary, SETPAGE will issue a SETXP request and empty the stack of page requests. Multiple page requests must refer to contiguous ascending VM addresses, so such a check is made before each new entry is added to the stack. If "unprocessed-by-loader" marking is not wanted for a page, a flag is put in the page's SETXP stack entry to suppress UPL bit setting by SETXP.

Note: The following are control blocks used by SETPAGE:

CHADCB - Data Control Block (VPAM for VSP member)

RHD - RESTBL (Relative External Storage Correspondence Table) header

DHD - DCB header (in RESTBL)

MHD - Member header (in RESTBL)

EPE - External Page Entry (in RESTBL)

PVT - Public/Private Volume Table

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
MAP SEARCH	Locate CSD of control section containing page to be relocated.	VMA of Page.	CSD address.
FIX	Process external REF RLD and internal REF RLD.	Modifier count, Pointer to first modifier, Page address.	
FREEMAIN	Release scratch pages.	Address and number of pages.	

PAGE RELOCATION is called by the task monitor whenever a "page unavailable" interrupt occurs, and the referenced page is also "unprocessed by loader" (see Chart BC and Figure 11).

Attributes: Privileged, public, system, reenterable.

Restrictions: Entrance by type-I linkage is restricted to the task monitor.

Entries: The input parameter to PAGE RELOCATION is in GR1 and is the address of a cell that contains the virtual storage address that caused the interrupt. This address is used as an input argument to MAP SEARCH to locate the CSD of the control section that contains this address.

Exits: Normal only, no return code.

Operation: Once the CSD is located by MAP SEARCH, the base address of the control section is extracted from the CSD and subtracted from the argument address. This difference is shifted right 12 places to obtain the page number of the referenced page, relative to the first virtual storage page of the control section. This page number is used as an index into the VMPT. The text page number is extracted from the VMPT entry and used as a relative index into the external RLD to locate the correct modifier pointer. The pointer for the subject page is obtained and the count extracted; then the VMA of the first modifier is computed from the pointer. FIX is called to process the external modifiers. The internal RLD is now processed in a fashion parallel to that for the external RLD. FIX is called again for the internal modifiers.

For example, PAGE RELOCATION is entered with an argument VMA = 8A32E. MAP SEARCH is entered and returns with a pointer to the related CSD from which the base address of the CSECT is extracted = 88000. The difference is computed, 232E, yielding a relative virtual page number equal to 2. Assume that the VMPT for this CSECT looks like:

F	F	F	F	0	0	0	0
0	0	0	1	F	F	F	F

The relative text page number is extracted from VMPT entry #2, giving 0001. The number 1 is a "word" index into both the external and internal RLDs; that is, the index must be multiplied by 4 to compute the offset in bytes from the origin of each of the RLDs to the modifier pointer for the argument page.

When privage control section packing is specified, PAGE RELOCATION is aware that some of the text pages to be relocated may not have been loaded into the page.

Any privage page that is being packed into is set unprocessed by the loader. The actual movement into its proper place in virtual storage takes place when the page is referenced. At this time all control sections to be packed into the page are read into a scratch page of virtual storage and transferred to their assigned virtual storage address in the packed page.

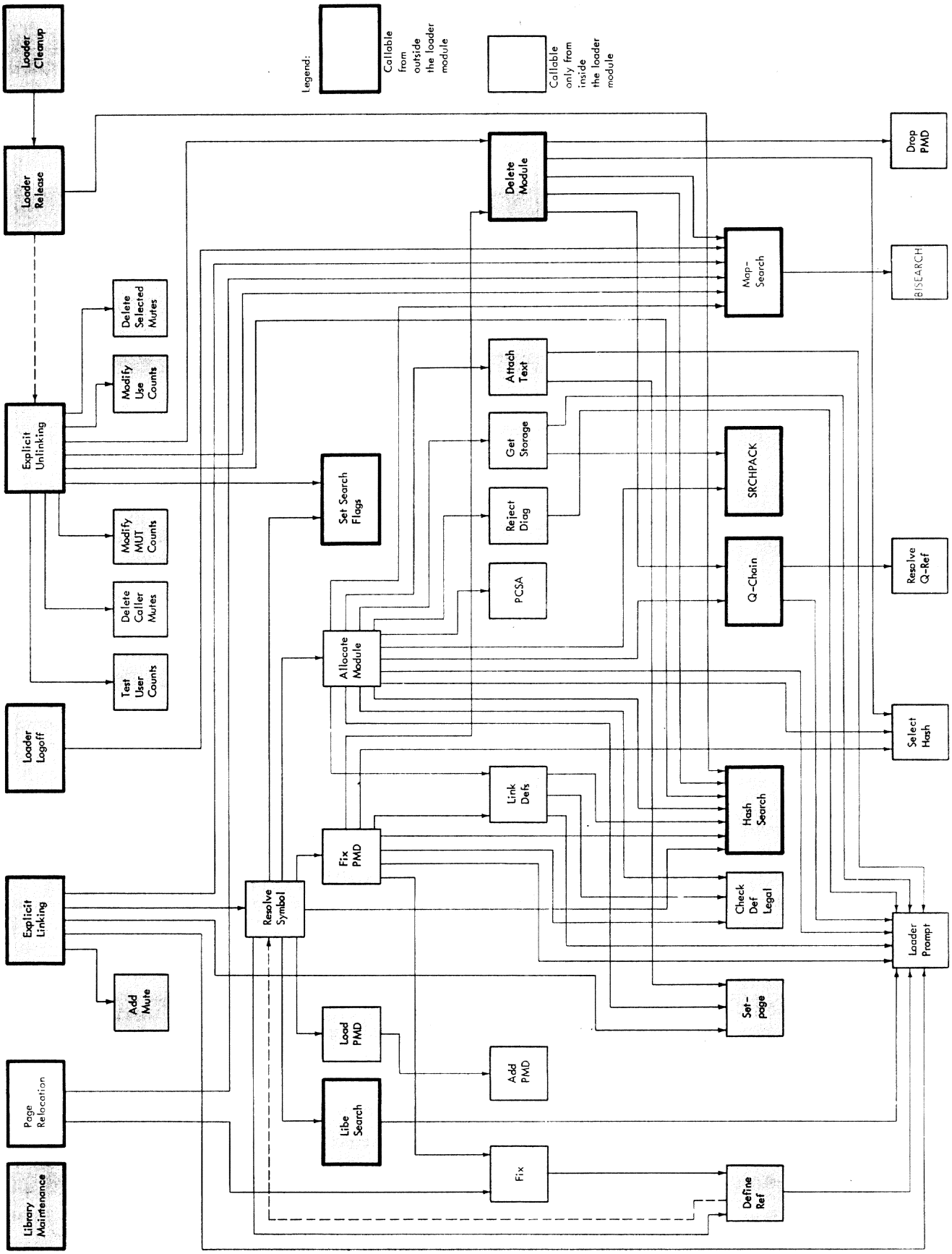


Figure 11. Page Relocation

When the relocation entrance of the loader is called, the low-order 12 bits of the referenced virtual storage address are cleared and MAPSEARCH is called to locate the CSD of the first control section assigned to the page. The adcons in the page are then relocated in the conventional manner.

Next, the packing table is searched (by storage key) for a host entry for the referenced page. If a host entry exists for this page, all of the remaining control section text is processed, one CSD at a time, by processing each subsequent symbiont entry. The text is moved from the scratch page to the packed page and its adcons are relocated.

Relocation is complete when the last symbiont entry has been processed or, if no packing entry exists, when the referenced page has been relocated.

Each control section is packed before any scratch pages are released. Contiguous pages of storage are released in a block with a single call to FREEMAIN. Then the host and symbiont entries (if they exist) are deleted and linked back into the available space table to prevent later packing into the page.

Comments: The techniques used in processing the RLDs are discussed in the following paragraphs.

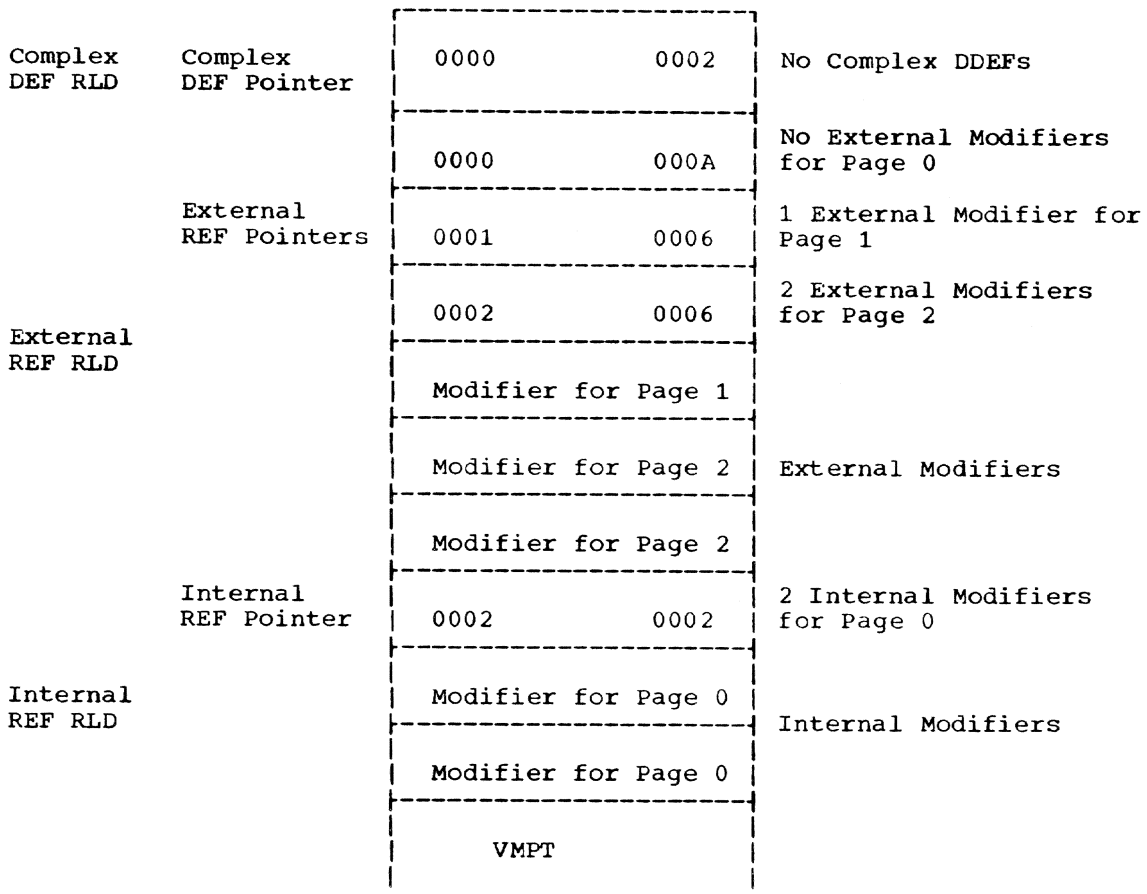
Each RLD is divided into two parts, the first of which contains a set of pointers, the second a set of modifiers. The RLD is organized by related control section text page. Modifiers exist in the RLD only for those text pages that contain adcons requiring relocation. A pointer exists for each page of the control section's text, up to and including the last page that contains a relocatable adcon. The RLD pointers are in linear order by relative text page number -- the first pointer is for text page 0; the second, for text page 1; etc.

Each pointer has two parts. The upper half contains the modifier count for the related page; the lower half contains a pointer (relative to this halfword itself) to the first of the group of modifiers for this page. If a pointer has a count field of zero, the pointer field will point to the location where the modifier group would have been located had there been any modifiers. The modifiers for each RLD are packed in such a way that the first modifier is in the word immediately following the last pointer. Furthermore, the last modifier for the complex DEF RLD immediately precedes the first pointer of the external REF RLD, and the last modifier for the external REF RLD immediately precedes the first pointer of the internal REF RLD. The last modifier of the internal REF RLD immediately precedes the virtual memory page table.

Given the origin of any RLD, the end of the RLD is located as follows: The first RLD pointer contains a relative pointer to the first modifier, which immediately follows the last pointer. Now the contents of the lower half of the first pointer are added to the location of the first pointer. This computes the location of the lower half of the last pointer. (Recall that the pointer is relative to the lower half of the pointer word and, therefore, adding the pointer to the location of the upper half of the pointer -- as is done here -- produces the location that precedes the modifier by two bytes. In the case of the first modifier, this locates the middle of the last pointer.) Now the modifier count is obtained from the upper half of the last pointer. (An index of -2 is used to back up from the lower half of this last pointer to obtain the modifier count from the upper half.) The modifier count is multiplied by four (the size in bytes of each modifier) to obtain the size of the last group of modifiers. Now it remains to compute the origin of this last group of modifiers by adding the location of the lower half of the last pointer (already computed above) to the contents of the lower half of this same pointer. The size of the group is added to this computed origin, and the sum is the location of the first byte past the end of the RLD.

The RLD modifiers are discussed more fully in Appendix B and in the description of the routine FIX.

A sample RLD:



EXPLICIT UNLINK (CZCDU1)

Routines Called in Pass 1	Purpose of Call	Parameters	
		In	Out
MAP SEARCH	Locate CSD of control section containing DELETE adcon group.	VMA of adcon group.	CSD address.
SET SEARCH FLAGS	Select hash table pointer for looking up argument symbol in DELETE adcon group.	Argument symbol, CSD pointer.	Hash table pointer.
HASH SEARCH	Look up argument symbol in selected hash chain.	Symbol name, hash table pointer.	Pointer to found DEF entry in TDY if found, else zero.
PRMPT	Unloader diagnostics.	Pointer to parameter string.	
DELETE CALLER MUTES	Collapse BABY chain for primary deletion candidate and rearm all explicit CALL/LOAD adcon groups.	PMD address of primary deletion candidate.	
MODIFY MUT COUNTS	Identify modules explicitly referenced by deletion candidates, decrement their MUT counts, and add referenced modules to the candidate list.	Candidate PMD address.	

In response to a user's DELETE macro instruction, or a call from LOADER RELEASE, explicit unlink attempts to remove from the task the module defining the named symbol and possibly subordinate modules as well (see Chart AM and Figure 12).

Attributes: Privileged, public, system, reenterable.

Restrictions: Entrance by type-I linkage is restricted to the task monitor.

Entries: EXPLICIT UNLINK is entered with GR1 pointing to a word that contains the virtual storage location of the DELETE adcon group, execution of which caused the task monitor to be entered. This adcon group has the format:

```
                DS   0F
CHD&SYSNDX      SVC  123          SVC for unloading
                DC   CL8'name'    Module name (or
                                    alias) to be
                                    unloaded
                DC   H'C3C4'      Unload options and
                                    return code
```

Exits:

0 = Normal

4 = Symbol to be unloaded not found (accompanied by diagnostic #16)

8 = Module not deleted because of outstanding references (accompanied by diagnostic #17)

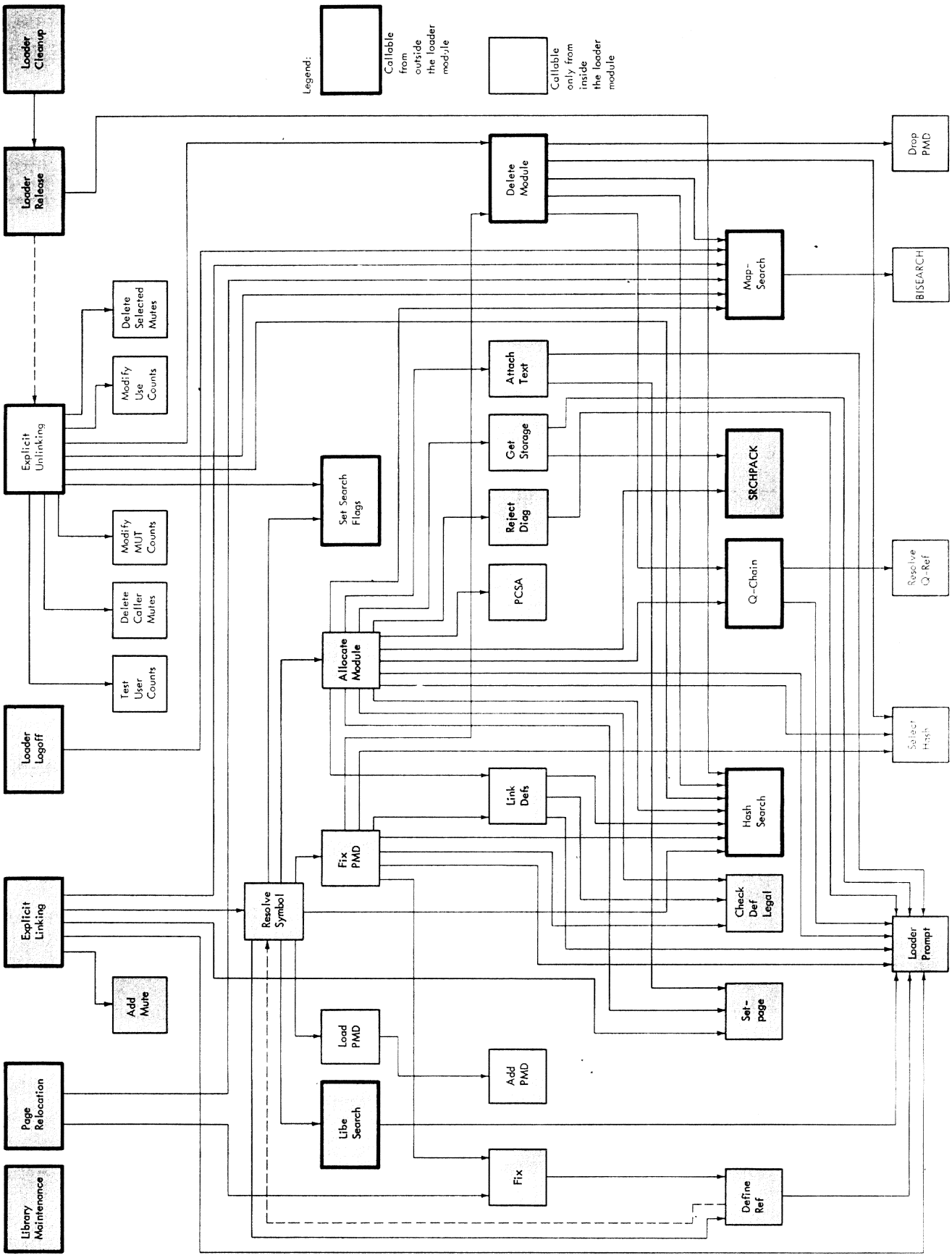


Figure 12. Explicit Unlinking

Operation: UNLINK's first action is to locate the CSD of the control section that contains the DELETE adcon group by calling MAP SEARCH (CZCDL5) in the loader module CZCDL with the adcon group address. Next SET SEARCH FLAGS (CZCDL6), also in the loader module, is called to set the hash table pointer for HASH SEARCH's eventual use in looking up the argument symbol. The high-order bit of the C3 option byte is used as a transpose-search flag, similar to the way in which the C1 option byte high-order bit is used in the loader processing of the LOAD/CALL adcon group. If this bit is set in the DELETE adcon group, UNLINK will set the transpose flag for SET SEARCH FLAGS. SET SEARCH FLAGS will return to UNLINK a hash table pointer (and a library index, which will be ignored). Now the loader routine HASH SEARCH (CZCDL2) is called to look up the argument symbol in the selected hash chain. If the symbol is not found, UNLINK will issue a diagnostic, set a return code of 4 in the C4 byte, and return to the task monitor with GR15 also set to 4.

If the symbol is found, UNLINK locates the containing module by using the defining DEF's CSD link to locate the containing CSD whose PMD link will locate the defining PMD preface. This PMD address is passed to DELETE CALLER MUTES (CGCDB), which will trace the BABY chain from the defining PMD preface to delete each MUT entry in the chain, rearm the original CALL/LOAD adcon group, and return the deleted MUTE to the MUT available-space chain.

Further processing by UNLINK is executed in four passes on a list of modules that are candidates for deletion at some time or other. This "candidate list" is a linear list of PMD preface addresses and is headed by the module defining the argument symbol name in the DELETE group. Construction of this list is under control of the low-order bit of the C3 option byte. If this "named-only" bit is set, the module defining module becomes the "primary" candidate, and "secondary" candidates may not be added to the list. A secondary candidate is one that either (a) contains a DEF that defines a REF in the primary candidate or another secondary candidate, or (b) has been explicitly called or loaded by the primary candidate or another secondary candidate. The module containing the DELETE adcon group is the only module in the task unconditionally proscribed from deletion candidacy (to prevent a module from unloading itself).

UNLINK pass 1 constructs the candidate list. Symbolic general register RN is initialized to zero and contains the relative displacement from the top of the list of the last module to have been entered. Symbolic general register RM is also initialized to zero and contains the relative pointer to the list member currently being processed in each of the four passes. The list is constructed by calling the two routines MODIFY MUT COUNTS (CGCDA) and MODIFY USE COUNTS (CGCDD) for the current candidate.

MODIFY MUT COUNTS traces the candidate's PAPA chain; that is, the chain of MUT entries describing each explicit CALL or LOAD made by the module. (This chain is constructed during EXPLICIT LINKING.) The MUT count in the PMD preface of each module explicitly referenced by the current candidate is decremented by one, and the referenced module becomes a secondary candidate with RN incremented to reflect the list addition.

Similarly, MODIFY USE COUNTS processes the REF table in each CSD of the current module. During EXPLICIT LINKING, the CSD link in each REF entry is set to point to the CSD containing the defining DEF (except in the case of undefined REFs whose CSD link is set to point to the CSD containing the REF itself). Now each REF's CSD link is used to locate the defining CSD whereupon the "use count" field is decremented by one, and the containing module is added to the candidate list as a secondary candidate. Again, RN is incremented to reflect the addition of the candidate.

Both MODIFY MUT COUNTS and MODIFY USE COUNTS set a flag in each candidate's PMD preface, which is checked prior to entering a new candidate, so as to preclude duplicating candidacy. These two routines are called for each candidate placed on the list until all candidates have been processed. RM will eventually catch up with RN, signaling the end of pass 1. Note that if the named-only option is in force, these two routines will refuse to post secondary deletion candidates on the list.

Pass 2 is a more complex loop designed to check the validity of each deletion candidate. The conditions favoring successful candidates are:

1. The primary candidate will be unloaded as long as there are no outstanding implicit references to any of the module's control sections. (There are no more explicit references to the primary candidate because DELETE CALLER MUTES has eliminated the BABY chain and rearmed the adcon groups.)

2. Secondary candidates will be unloaded so long as no implicit or explicit references to the candidate remain.

3. Any module may be unloaded except the one containing the DELETE adcon group being processed.

Routines Called in Passes 2-4	Purpose of Call	Parameters	
		In	Out
MODIFY USE COUNTS	Identify modules implicitly referenced by deletion candidates, decrement their CSD use counts, and add referenced modules to the candidate list.	Candidate PMD address.	
TEST USER COUNTS	Check each candidate for zero MUT count and zero use counts.	Candidate PMD address.	Flag indicating all zero counts, or one count not zero.
MODIFY MUT COUNTS	Increment MUT counts in modules explicitly referenced by a disqualified candidate.	Candidate PMD address.	
MODIFY USE COUNTS	Increment use counts in CSDs of modules implicitly referenced by a disqualified candidate.	Candidate PMD address.	
DELETE SELECTED MUTES	Collapse PAPA chain for each successful candidate.	Candidate PMD address.	
DELETE MODULE	Remove PMD and text of each successful candidate from virtual storage.	Candidate PMD address.	
PCS UNLOAD	Process PCS tables for unloaded modules.	Candidate PMD address.	

The first two conditions are examined in pass 2 by the routine TEST USER COUNTS (CGCDE), which is called for each candidate. TEST USER COUNTS tests for zero the MUT count field in the PMD preface and the use count field in each of the candidate's nonrejected CSDs for zero. If any of these fields is nonzero, a reference exists from some module not a candidate, and UNLINK proceeds to erase the candidate by:

1. Calling MODIFY MUT COUNTS to process the candidate's PAPA chain by examining each MUT entry in the chain to locate the referenced PMD in whose preface the MUT count field is now incremented by one.

2. Calling MODIFY USE COUNTS to process each REF entry in each of the candidate's CSDs to locate the defining CSD in which the use count field is incremented by one.

3. Erasing the candidate flag in the PMD preface of the disqualified candidate.

4. Marking the candidate "disqualified" in the candidate list by setting a low-order 1 bit in the list. (Since the list consists of PMD preface addresses which are fullword address, the low-order bit is a convenient method of flagging disqualified candidates.)

Each time a candidate is disqualified, a flag is set at the bottom of the pass 2 loop. Restoring the MUT and use counts during disqualification makes possible the disqualification of other candidates on the list. Therefore, after the candidate list has been completely processed, this flag is checked. If it has been set, indicating a disqualification, the flag is reset and the list processed once again to identify any newly disqualified candidates. This process is repeated until a pass is made on the list that results in no new disqualifications. At this point pass 2 is complete, and all the candidates remaining on the list will be unloaded.

Pass 3 is a simple pass in which DELETE SELECTED MUTES (CGCDC) is called for each candidate to trace the PAPA chain of the module, deleting all the MUT entries, and returning the MUTEs to the MUT available space chain. Since each module explicitly referenced by the candidate is now a suc-

cessful candidate, the MUTEs serve no further function.

Pass 4 is the unloading pass in which DELETE MODULE (CZCDU2) is called for each candidate. DELETE MODULE performs the following functions:

1. Deletes all nonrejected DEFs in each nonrejected control section from the appropriate hash chain.

2. Frees for reassignment the DXD areas not referenced by modules that are still loaded.

3. Frees virtual storage for each nonrejected control section.

4. Deletes the MAP entry associated with each nonrejected control section.

5. Deletes the candidate's PMD from the TDY.

Pass 4 is made separate from pass 3 because during the processing of the PAPA chains in pass 3, the PMD preface of each candidate will be referenced in the process of collapsing the chain. Since the preface must still be intact during this process, the deletion is placed in a fourth pass.

Normal exit is made after pass 4 with the C4 byte and GR15 set to zero. Note that if after pass 3 it is discovered that the primary candidate has been disqualified, pass 4 is bypassed. It is impossible to have qualified secondary deletion candidates if the primary candidate is disqualified; hence, no unloading can take place. In this event, a diagnostic is issued, and GR15 and the C4 byte are set to 8.

Notes:

1. Modules loaded into initial virtual storage (IVM) by STARTUP will have had their MUT count fields set to an arbitrarily high number by STARTUP to preclude the unloading of any IVM module.

2. A corollary to the unloading algorithm described in this section is that only modules explicitly loaded (by virtue of an explicit CALL or LOAD macro instruction) may ever be successful primary deletion candidates.

Comment: Figure 13 shows the general flow of explicit unlinking. It is separated into functional segments, rather than specific routines, to aid the reader in obtaining an overview of the unlinking process.

See Appendix B for a detailed account of the MUT structure.

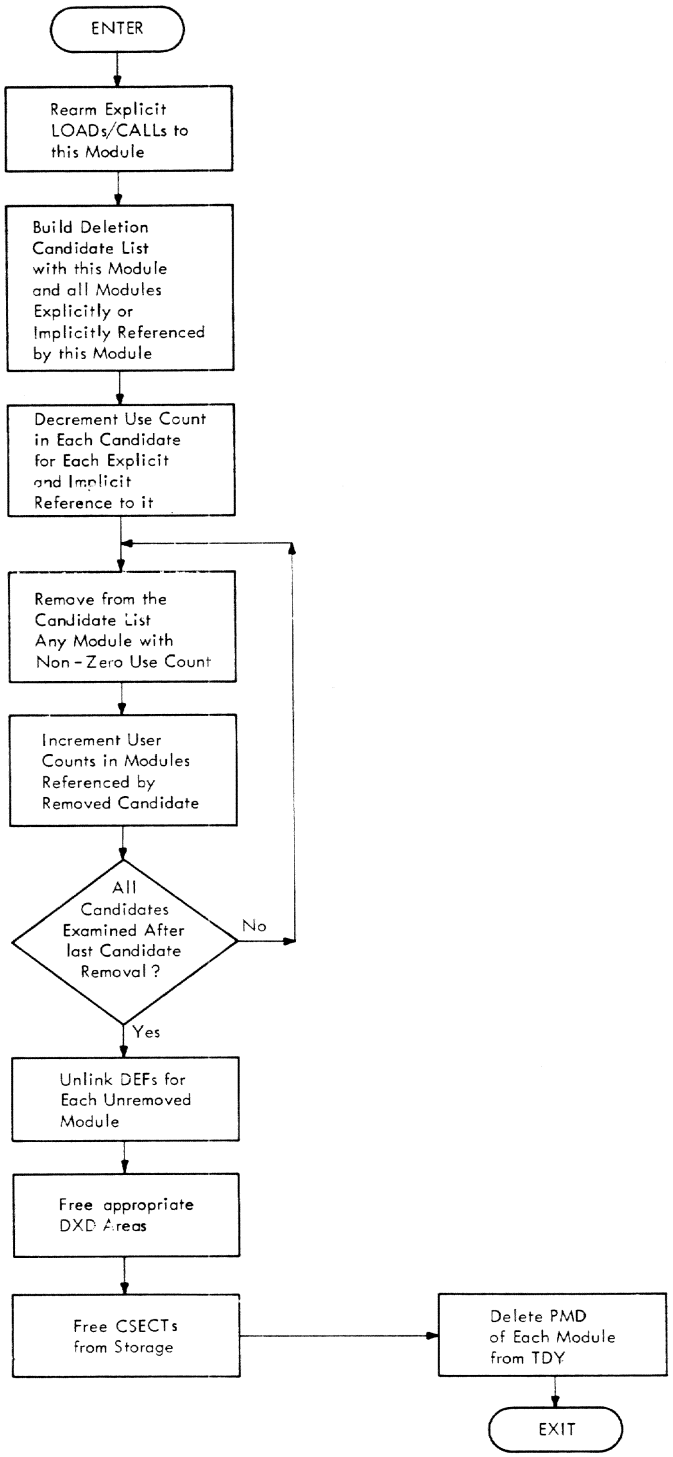


Figure 13. Functional Diagram of Explicit Unlinking

DELETE CALLER MUTES (CGCDB)

For a particular module, all module usage table (MUT) entries for explicit CALLs on that module are deleted (see Chart AH).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to unloader module, not available to other system components.

Entries: On entrance, GR1 contains the address of the PMD for a specified module (subject PMD) as an input parameter.

Routines Called: None.

Exits: Normal only, no return code.

Operation: From this PMD preface, the BABY chain head is fetched that points to the first MUT entry in the chain. This MUT entry is then deleted, as follows:

1. It is removed from its BABY chain by relinking the chain in the forward direction so that the BABY chain head in the subject PMD is pointing to the next MUTE in the chain (or is zero).

2. It is removed from its PAPA chain by relinking the PAPA chain bi-directionally.

3. The space occupied by the MUT entry is returned to the MUT available space chain.

4. In the subject PMD preface, the MUT count field is zeroed.

5. The SVC within the adcon group that originally effected the explicit linkage is rearmed with the DLINK SVC. Additionally, the V-con portion of this adcon group is set to point to the DLINK. (The address of the adcon group is contained in the MUTE.)

Any additional MUT entries in the subject PMD's BABY chain are similarly deleted. The processing stops when the BABY chain head in the argument PMD preface is zero.

See Appendix B for a description of MUT.

MODIFY MUT COUNTS (CGCDA)

The MUT count field in the PMD preface of each module explicitly referenced by the argument module is decremented or incremented by one, according to a parameter (see Chart BA).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to unloader module; not available to other system components.

Entries: On entrance to MODIFY MUT COUNTS, GR1 contains the address of the argument PMD preface, and GR2 contains a function code. The function code is 0 for decrement and 1 for increment.

Routines Called: None.

Exits: Normal only, no return code.

Operation: The PAPA chain head in the argument PMD preface begins the chain of MUT entries that describe each of the explicit references made by the module. Each MUTE in this chain is examined in turn, and the PMD preface address of the referenced module is extracted. Within this referenced PMD preface, the "MUT count" field is either incremented or decremented by one according to GR2.

On the decrement function, MODIFY MUT COUNTS will check to see if the named-only option is in force, in which case the next MUTE is examined, as in the increment case. If the named-only option is not in force, the referenced PMD preface is checked for the setting of the deletion candidate flag. If this flag is not set, the PMD preface address is added to the candidate list, the candidate flag is set in the PMD preface, and symbolic general register RN is incremented by 4 to point to the new candidate's relative position on the list. If the deletion candidate flag is found to be set, MODIFY MUT COUNTS will not again add the module to the candidate list.

MODIFY USE COUNTS (CGCDD)

For every REF in a specified PMD, the use count in the CSD that contains the referenced DEF is incremented or decremented according to an input parameter (see Chart BB).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to unloader module; not available to other system components.

Entries: On entrance to MODIFY USE COUNTS, GR1 contains the address of the argument PMD preface, and GR2 contains a function code. The function code is 0 for decrement and 1 for increment.

Routines Called: None.

Exits: Normal only, no return code.

Operation: Each REF in each nonrejected CSD is examined. The CSD link in each REF entry will point to the CSD of the DEF entry that defines the REF. (Undefined REF's CSD links will point to the CSD containing the REF entry.) The user count

field in the defining CSD is now decremented or incremented by one according to GR2.

On the decrement function, MODIFY USE COUNTS will check to see if the named-only option is in force, in which case the next REF is examined, as in the increment case. If the named-only option is not in force, the PMD containing the defining CSD is checked for the setting of the deletion candidate flag. If this flag is not set, the PMD preface address is added to the candidate list, the candidate flag is set in the PMD preface, and symbolic general register RN is incremented by 4 to point to the new candidate's relative position on the list. If the deletion candidate flag is found to be set, MODIFY USE COUNTS will not again add the module to the candidate list.

TEST USER COUNTS (CGCDE)

A specified PMD is tested to discover if there are any explicit CALLs or implicit references to it (see Chart BM).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to unloader module; not available to other system components.

Entries: On entrance, GR1 contains the address of the argument PMD preface.

Routines Called: None.

Exits: GR2 is zero if all counts test zero; GR2 is nonzero if some count tests nonzero.

Operation: For the argument PMD, the MUT count field in the PMD preface and the user count field in each nonrejected CSD heading is tested for nonzero. On exit, GR2 will contain a condition code; zero indicates that all count fields tested zero; and a positive number indicates that some field tested nonzero. On a nonzero test, TEST USER COUNTS exits immediately.

DELETE SELECTED MUTES (CGCDC)

For a particular module, all module usage table (MUT) entries for explicit CALLs by that module are deleted (see Chart AJ).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to unloader module; not available to other system components.

Entries: On entrance, GR1 will contain the address of the PMD for a specified module (subject PMD) as an input parameter.

Routines Called: None.

Exits: Normal only, no return code.

Operation: From the PMD preface, the PAPA chain head is fetched that points to the first MUT entry in the chain. This MUT entry is then deleted as follows:

1. It is removed from its PAPA chain by relinking the chain in the forward direction so that the PAPA chain head in the subject PMD is pointing to the next MUTE in the chain (or is zero).

2. It is removed from its BABY chain by relinking the BABY chain bi-directionally.

3. The space occupied by the MUT entry is returned to the MUT available space chain.

Additional MUT entries in the subject PMD's PAPA chain are similarly deleted. The processing stops when the head of the PAPA chain in the argument PMD preface is zero.

See Appendix B for a description of MUT.

DELETE MODULE (CZCDU2)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
SELECT HASH	Select hash table for DEF deletion.	CSD address.	Hash table pointer.
HASH SEARCH	Delete DEFs from selected hash table.	Hash table pointer, DEF entry pointer.	
Q-CHAIN	Delete Q-REFs from selected hash chains.	Address of CSD, function code (delete).	
MAP SEARCH	Delete MAP entry for each control section.	Control section address.	
SRCHSDST	Close control section group member entry in SDST.	Member name.	Code indicating whether control section group is still being shared by other tasks.
FREEMAIN	Free private text pages or free public text pages if no other task using.	Page count, address.	
DISCONNECT	Disconnect task from control section group shared pages if other tasks still using.	SPT #, relative page number.	
DROP PMD	Delete PMD from TDY.	PMD address.	

A specified module and the table entries that describe it are deleted from the task (see Chart AI).

Attributes: Privileged, public, system, reenterable.

Restrictions: Accepts type-I linkage only from other privileged system components.

Entries: DELETE MODULE is entered by type-I linkage with GR1 pointing to a parameter that contains the address of the PMD preface of the module to be deleted.

Exits: Normal only, no return code.

Operation: The first function is the deletion of the module name DEF entry, if the DEF was not rejected during loading. SELECT HASH is called with the CSD address obtained from the module name DEF CSD line; then HASH SEARCH is called to delete the DEF entry from the selected hash chain.

Now each CSD is processed as follows:

Private Control Sections:

1. The MAP table entry is deleted by calling MAP SEARCH with the delete option. Note that only control sections of nonzero text length have associated MAP entries; hence, text length is checked prior to the MAP SEARCH call.
2. Q-CHAIN is called to reclaim storage areas that were reserved by DXD (or DSECT) and CXD instructions but are no longer referenced by loaded modules.
3. SELECT HASH is called to select the hash chain in which all the DEFs for the current CSD are posted.
4. Each relocatable, absolute, and complex DEF is processed, and those with nonzero CSD links are deleted from the hash chain by a call on HASH SEARCH. (Those with zero CSD links were rejected during the loading processed and never posted.)
5. Rejected private CSDs are not processed, since no DEFs may be posted from rejected CSDs, and no virtual storage is allocated.
6. Control section text pages, if any, are freed by calling FREEMAIN. If control section packing is specified

for private control sections, all but the last page are released unconditionally. Then a search is made of the MAP table for another control section with the last page address. If other control sections exist on the last page, no FREEMAIN is done, and the MAP entry for the unloaded control section is deleted.

Public Control Sections:

1-4 Steps 1 through 4, above, are the same for public control sections.

5. Following the DEF deletion process, the public name bit in the CSD attributes halfword is tested. If the bit is not set, processing skips to the next CSD in the module. If this bit is set, the item "SDST name" is set to the current control section name, except the first time through this path for each module. The first time through, SDST name is set to the module name. The public name bit is set in the first named CSD of each group of public CSDs of like attributes by the EXPLICIT LINKING routine ALLOCATE MODULE (CGCCA). This bit tells DELETE MODULE that the name of this control section is the name carried in the SDST member entry that describes the public control section group. (Remember the exception: the first public control section group carries the module name in the SDST member entry.)

6. Now SRCHSDST (CZCQE) is called to close out the SDST member entry whose name is the same as SDST name, and whose parent data set SDST entry bears the same name as the data set name in the JFCB from which the current module was loaded. (Each PMD preface contains the address of the JFCB describing the containing data set.) If SRCHSDST returns with a code indicating that the user count for this entry is nonzero, DELETE MODULE merely calls DISCONNECT to disconnect the task from the shared pages. If SRCHSDST returns indicating a zero use count -- meaning that there are no more shared users of the storage -- DELETE MODULE goes through a loop, processinng each CSD, and looking for all those whose attributes are an exact match of the current CSD. The text pages for those CSDs whose attributes match are freed by calling FREEMAIN.

   If a packed control section in the group overlaps into the next page of virtual storage, storage is allocated by subgroups. The control section overlapping the page is the first control section in the next subgroup. If an overlap is detected in the process of searching each CSD for the attributes that match the "current" CSD, the search is discontinued and the text pages for the CSDs in the group are freed by calling FREEMAIN.

7. Storage is allocated to variable-length public control sections individually, which requires each control section's CSD to have the public name attribute bit on. When SRCHSDST returns, indicating zero use count, variable-length control section text pages are freed without going through the attribute-matching loop described above.

The conditions under which packed public control sections are released are more complex. When SRCHSDST is called to close out a member entry, it must decrement the user count in the found entry and in the host SDST entry, if the found entry is a symbiont SDST entry. SRCHSDST takes the zero user count exit only if both the symbiont and host SDST entries have gone to zero. (Note that the host SDST entry can only go to zero if all symbiont user counts have gone to zero as well as user count for the control section group represented by the host entry itself.)

In the event that SRCHSDST returns with user count equal to zero, the unloader knows that it is free to release the shared storage via FREEMAIN. In this case, the VST is searched for an entry for the last page of the control section group represented by the host SDST entry. If none is found, there is no action. If one is found, this entry is deleted from the packing table.

DROP PMD (CGCCO)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
FREEMAIN	Free storage occupied by collapsed PMD group.	Page address, page count.	

DROP PMD is called to release a PMD from a PMD group (see Chart AK).

Attributes: Privileged, public, system, reenterable.

Restrictions: Internal to unloader module; not available to other system components.

Entries: GR1 contains the input to DROP PMD, which is a pointer to the preface of the PMD to be released.

Exits: Normal only, no return code.

Operation: DROP PMD unlinks the current PMD from the group. First the TDTBLK field of the JFCB of the library from which the module was loaded is decremented, to indicate the unloading of a module. Since there are no back links, the forward links are traced through the circular chain until the PMD to be released is found, at which point the link from the previous PMD is changed to point to the PMD immediately following the released PMD.

Because of the circular chain structure, a collapsed PMD group is defined when a pointer to the next PMD points to itself, and this pointer will always be in the PMD group header. If the PMD group does not collapse, DROP PMD exits. If the group does collapse, DROP PMD relinks the PMD group chain, both forward and backward, and releases the space occupied by the PMD group through FREEMAIN.

LOADER LOGOFF (CZCCD1)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
SRCHSDST	Close control section group member entry in SDST.	Member name.	Code indicating whether any other shared users.
FREEMAIN	Free storage for control section groups with no other shared users.	Page count, page address.	
DISCONNECT	Disconnect current control section group shared pages when there remain other shared users.	SPT number, relative page number.	
MAPSEARCH	Find CSD for control section.	Virtual storage address.	Address of CSD for that control section.

LOADER LOGOFF (see Figure 14) is called at task end for public storage housekeeping to reflect that the current task is no longer a shared user (see Chart AW).

Attributes: Privileged, public, system, reenterable.

Restrictions: LOADER LOGOFF is a special purpose routine available to other privileged system service routines by type-I linkage, but not designed for general system service usage.

Entries: LOADER LOGOFF is entered by type-I linkage, with no parameters.

Exits: Normal only, no return code.

Operation: LOADER LOGOFF will examine each non-IVM PMD in each PMD group in the task dictionary (TDY). The public flag in each PMD preface is tested. If the flag is not set, the PMD has no public control sections, and the next PMD is examined. PMDs with the public flag set are further processed, a CSD at a time, as follows:

1. Public CSDs are checked for public name attribute bit on.

2. The item SDST name is set to the control section name for those CSDs with the public name bit on, except that the first time each module travels this path, the item SDST name is set to the module name. This algorithm is identical to the one used in the loader routines ALLOCATE MODULE and GET STORAGE in the allocation of public storage during EXPLICIT LINKING.

3. SRCHSDST (CZCQE) is called to close out the member entry in the SDST: (1) whose name matches public name, and (2) whose parent data set SDST entry matches the data set name in the JFCB that describes the data set from wich the current module was loaded. (The JFCB pointer is extracted from the PMD preface.) If SRCHSDST returns, indicating that the user count for this member is nonzero, LOADER LOGOFF merely calls DISCONNECT to disconnect the task from the shared pages used by the member. (These shared pages are all pages allocated for the control section group when it was first loaded.) If the user count went to zero on the close, LOADER LOGOFF executes a loop, processing each CSD. The CSDs whose attributes match the current CSD's will have their text length added to a running total. At the end of the loop, the total number of text pages from the control sections of matching attributes is computed, and the origin of this block of pages will be the origin of the first control section processed with these attributes. The block of pages is now freed through FREEMAIN, and processing of the module

proceeds to the next CSD with public name on.

If SRCHSDST returns a code of "data set does not exist," a SYSER is caused. Upon return from SYSER, processing continues normally.

The conditions under which packed public control sections are released are more complex. When SRCHSDST is called to close out a member entry, it must decrement the user count in the found entry and in the host SDST entry, if the found entry is a symbiont SDST entry. SRCHSDST takes the zero user count exit only if both the symbiont and host SDST entries have gone to zero. (Note that the host SDST entry can only go to zero if all symbiont user counts have gone to zero as well as user count for the control section group represented by the host entry itself.)

In the event that SRCHSDST returns with user count equal to zero, the loader knows that it is free to release the shared storage via FREEMAIN. In this case, the VST is searched for an entry for the last page of the control section group represented by the host SDST entry. If none is found, there is no action. If one is found, this entry is deleted from the packing table.

If a packed control section in the group overlaps into the next page of virtual storage, storage is allocated by subgroups. The control section overlapping the page is the first control section in the next subgroup. If an overlap is detected in the process of searching each CSD for the attributes that match the "current" CSD, the search is discontinued and the text pages for the CSDs in the subgroup are freed by calling FREEMAIN.

LOADER LOGOFF ceases its processing when the forward PMD group link is equal to the IVM pointer in the TDY header.

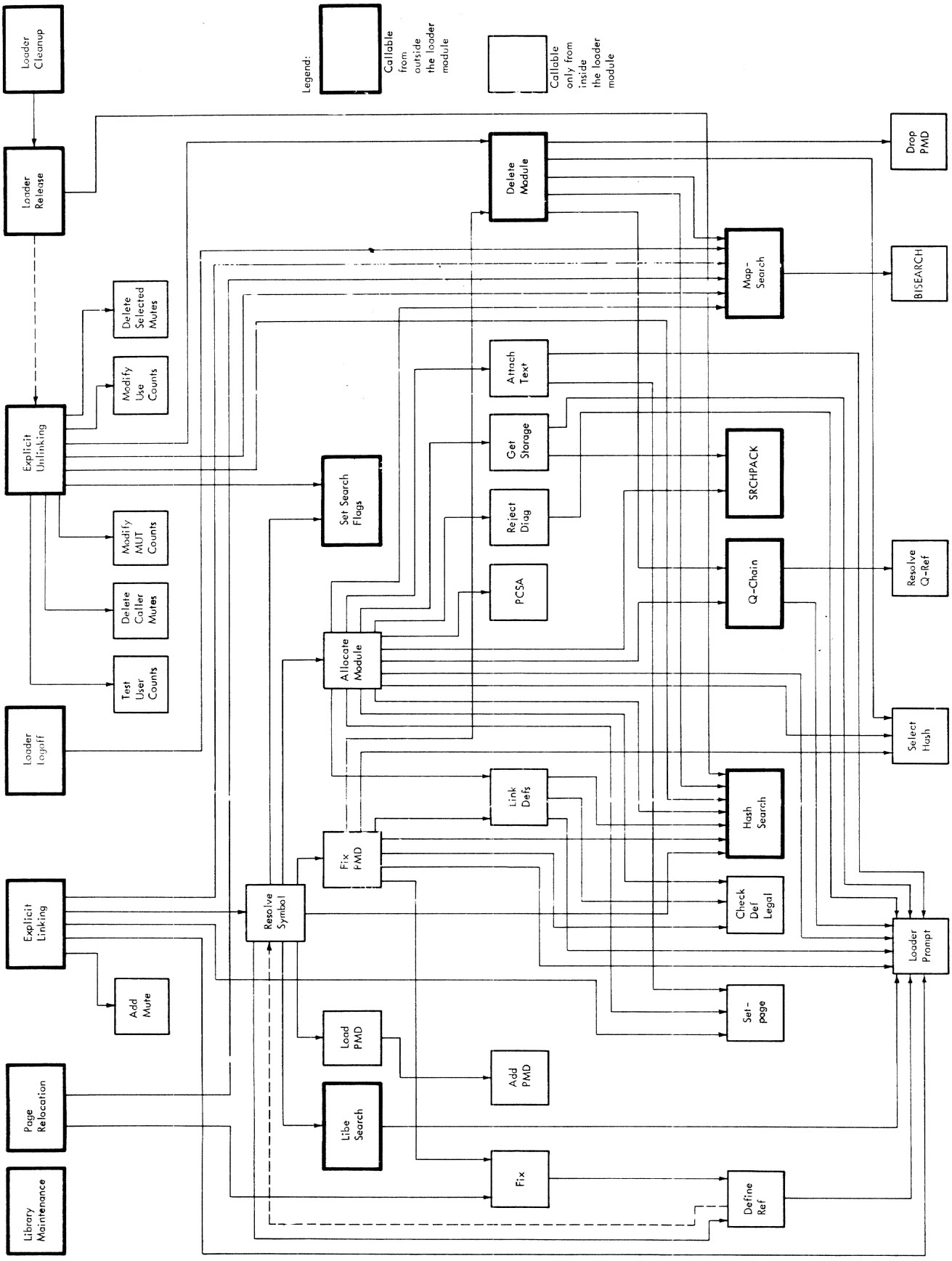


Figure 14. Loader Logoff

SECTION 6: LOADER RELEASE

LOADER RELEASE (CZCCD2)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
EXPLICIT UNLINKING	Unload modules from library to be released.	VMA of adcon group.	Code indicating whether module was unloaded.
HASH SEARCH	Look up module in Hash Table.	Module name.	Code indicating whether module was found.
PRMPT	Inform user module was not unloaded due to outstanding references.	Pointer to parameter string.	

LOADER RELEASE (see Figure 15) is called to unload from virtual storage all modules belonging to a library (remove them from the TDY) before a 'DDEF' is released (see Chart AX).

Attributes: Privileged, system, reenterable.

Restrictions: Accepts type-I linkage only from other privileged system components.

Entries: LOADER RELEASE is entered by type-I linkage, with GR1 pointing to the JFCB of the library to be released.

Exits: Return code in GR15:

0 = no errors found
4 = errors found

Operation: LOADER RELEASE searches through the TDY for all non-IVM modules with TDYJFC (JFCB pointer in PMD preface) equal to the parameter passed. Whenever one is found, an attempt is made to unload it by a call to EXPLICIT UNLINK (CZCDU1). If the attempt is not successful (indicated by an error return code), the module name is added to an internal list. When the search through the TDY is complete, a return code of zero is set if there were no error codes returned from EXPLICIT UNLINK. If there were errors, a return code of four is set and, if LOFOFF is not in process, HASH SEARCH is called to look up each module name in the list. The user receives a diagnostic message containing the module name for each module still not unloaded.

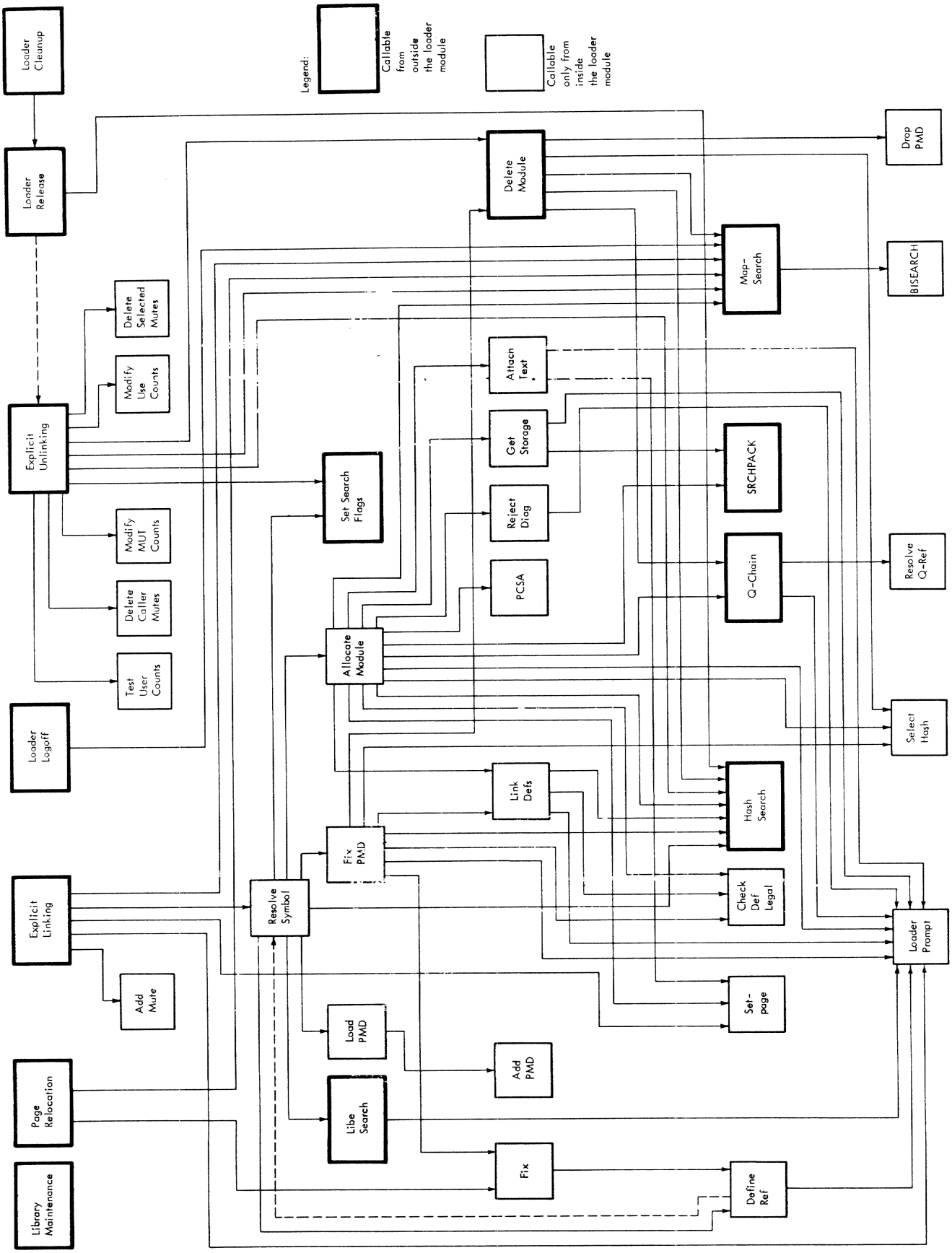


Figure 15. Loader Release

LOADER CLEANUP (CZCCD4)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
LOADER RELEASE	To unload all modules loaded from a JOBLIB.	Pointer to JFCB.	

LOADER CLEANUP (see Figure 16) is called by LOGOFF to unload all modules loaded during an express batch subtask so that the TDY is clean for the next subtask (see Chart AU).

Attributes: Privileged, public, system, reenterable.

Restrictions: LOADER CLEANUP is a privileged system service routine available only to other service routines by type-I linkage, but not designed for general use.

Entries: LOADER CLEANUP is entered by type-I linkage with no parameters.

Exits: Normal only, no return code.

Operation: LOADER CLEANUP picks up the ISAJLC pointer to the DCB of the last defined job library; the JFCB is covered by the address in the DCB header. The count of modules loaded (TDTBLK) is checked for zero; if nonzero, the address of the JFCB is passed as a parameter to LOADER RELEASE. Upon return from LOADER RELEASE, or if the count was zero, the DCB pointer is moved to cover the DCB pointed to by the forward pointer of the DCB header. If this pointer is nonzero, LOADER CLEANUP loops back to check the count; if it is zero LOADER CLEANUP returns to the caller.

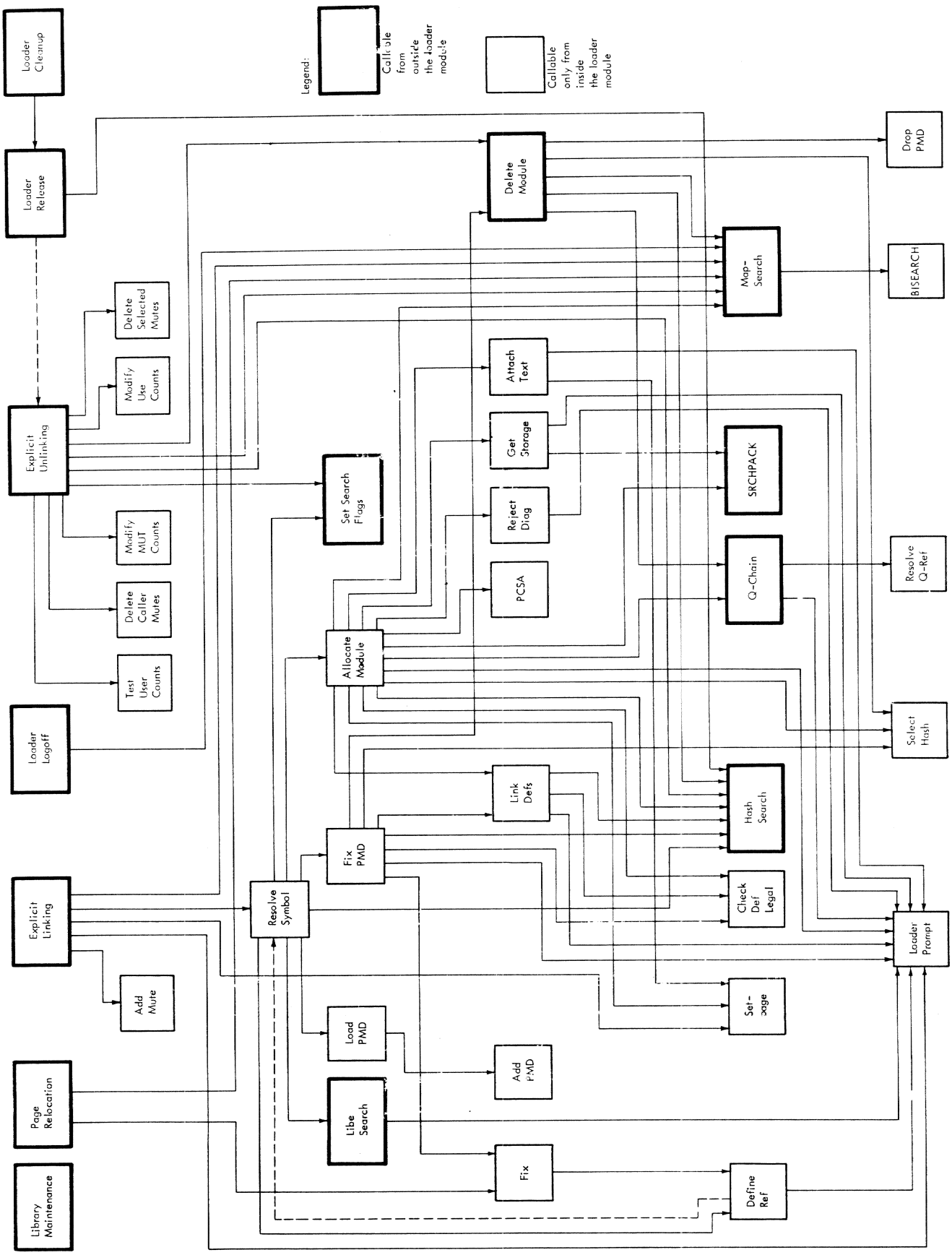


Figure 16. Loader Cleanup

LIBE MAINT (CZCDH)

ROUTINES CALLED	Purpose of Call	Parameters	
		In	Out
GETMAIN	Get storage to expand DCB chain.	Page count, storage class.	Address of page.
OPEN	Open new DCB.	DCB address.	
CLOSE	Close DCB.	DCB address.	
GATWR	Diagnostic when library to be closed not found.	Pointer to parameter string.	
SHARE (CZCFS1)	Mark catalog entry as shared.	Fully qualified data set name, number of sharers, list of sharers.	

LIBE MAINT (see Figure 17) is called during virtual memory task initialization to open SYSLIB for the task. It is called again by LOGON to open the user's private library (USERLIB), and again in response to any JOBLIB DDEF command the user has entered during the life of his task (to open DCBs for the newly defined libraries). LIBE MAINT may also be called at various times to close JOBLIBs opened earlier and will be called at LOGOFF to close USERLIB and SYSLIB (see Chart AR).

Attributes: Privileged, public, system, reenterable.

Restrictions: LIBE MAINT is a special purpose routine available to other privileged system service routines by type-I linkage, but not designed for general system service usage.

Entries: LIBE MAINT is entered by type-I linkage with GR1 pointing to a word that contains the address of the following list:

1. A pointer to the JFCB of the data set to be processed.

2. A function where 0 indicates "add" and 1 indicates "delete" the library from the chain.

Exits:

Normal: GR15=0
Error: GR15=4, library to be closed not found

Operation: LIBE MAINT maintains a chain of open DCBs, each preceded by a header. The chain defines the program job library hierarchy used by both the dynamic loader and linkage editor to locate symbols defined in program modules. The libraries are all in the form of partitioned data sets. LIBE MAINT's PSECT (CZCDHP) is coded with a small number of builtin DCBs. Each of these is preceded by a four-word header, the first word of which is used to chain these DCBs together in an available DCB chain. The DCBs are each coded with DSORG=VP, LRECL = 4096, and RECFM=U; that is, the format is virtual partitioned, records will be transferred a page at a time, and the record format is undefined (meaning records are merely single pages of hexadecimal data). This chain of DCBs resides in privileged storage, and therefore can be used only by privileged routines. Also maintained in LIBE MAINT's PSECT is a model DCB that is never opened and is used to generate new DCBs when the available DCB list is exhausted.

The open DCBs for a single task are chained together through headers. There are two pointers in the read-only half of the ISA which delimit this chain: ISAJLC points to word 0 of the header for the last-opened DCB in the chain, and ISASLP points to word 0 of the header of the first DCB opened for the task. Both the dynamic loader and LIBE MAINT assume that the first opened data set is, in fact, the SYSLIB data set; SYSLIB is defined for the loader as the first-opened data set, not one whose DDNAME is necessarily "SYSLIB."

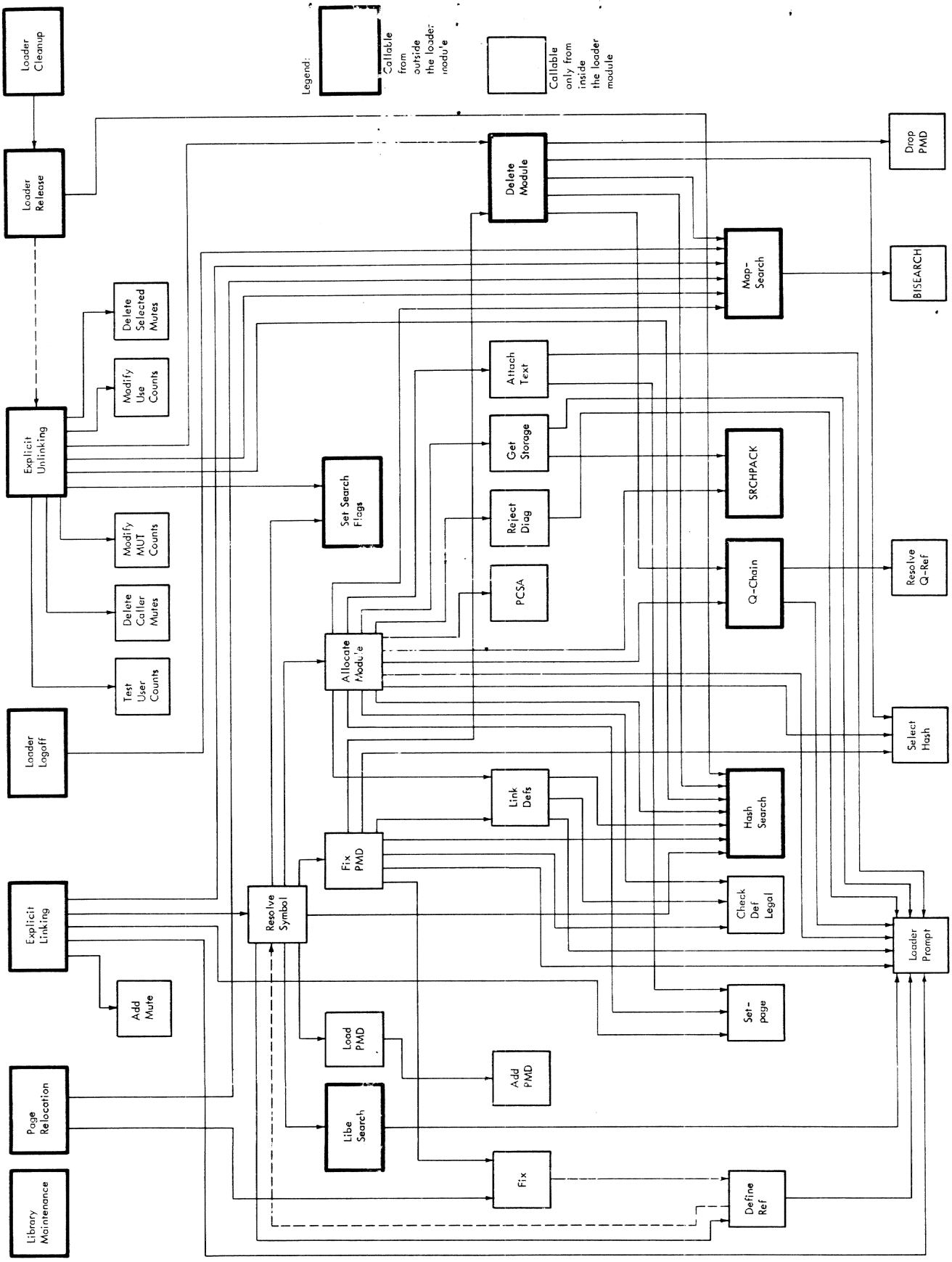


Figure 17. Library Maintenance

The DCB headers are chained bi-directionally through the first two words of the header. The forward link of the SYSLIB DCB header will be zero; the back link of the last-opened JOBLIB will point to ISAJLC. DCB headers are of the following format:

Forward Link: Pointer to Word 0 of Next DCB Header
Back Link: Pointer to Word 0 of Previous DCB Header
Pointer to JFCB for Data Set
Not Used

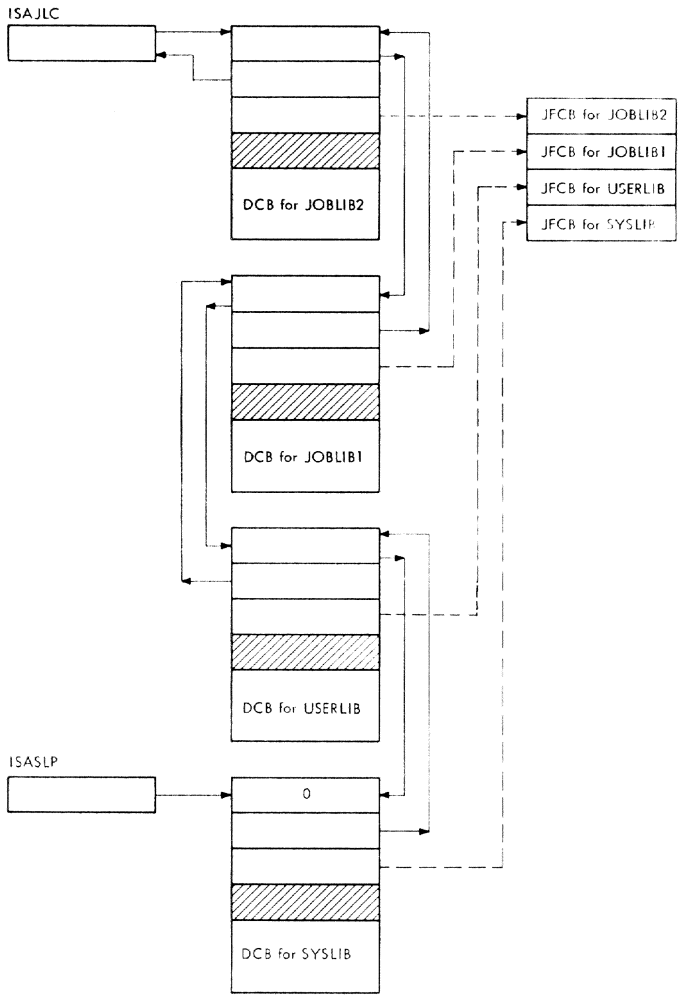
The Add Function: The DCB availability chain is checked: if the pointer is zero, indicating no available DCBs, one storage class C page is requested via GETMAIN. The model DCB with its header is copied into this until the page contains the maximum number of DCB/header blocks. The available space pointer is set to point to the first word of the new page (which is word 0 of the first header), and the remainder of the DCBs are chained together in the availability list through word 0 of their respective headers.

Now the first DCB is plucked from the availability list and chained into the head of the existing program library chain. That is, the forward link of the new header is set to the contents of ISAJLC, the new header back link is made to point to ISAJLC, and ISAJLC is set to point to the new header. The original contents of ISAJLC are checked for zero -- indicating the first add call on LIBE MAINT. In the first call, ISASLP is set to point to the new header, whose attached DCB is assumed to be the DCB for SYSLIB.

If ISAJLC was not zero, the back link of the previous head of the chain is set to point to word 0 of the new header.

The JFCB pointer -- parameter 2 -- is now placed in the DCB header, and an OPEN macro instruction is executed to open this new DCB, making the data set available to the loader and link editor. If the library opened is USERLIB and the data set is not shared, CLOSE is called. Then SHARE (CZCFS1) is called to mark the catalog as shared by the owner, and then OPEN is called again. This permits a user to have multiple tasks running simultaneously.

A typical chain might look like this:



The Delete Function: The chain is searched from the beginning, and each header is examined to locate that one whose JFCB pointer matches the first input parameter. If no match is found, a diagnostic is issued via GATWR and an error exit is taken. If a match is found, the DCB header is removed from the chain. The DCB with header is added to the head of the availability list, and a CLOSE macro instruction is executed on the removed DCB.

Exit on the add function is always made with GR15 set to zero. Normal exit on the delete function is made with GR15 set to zero; error exit on delete is with GR15 set to 4.

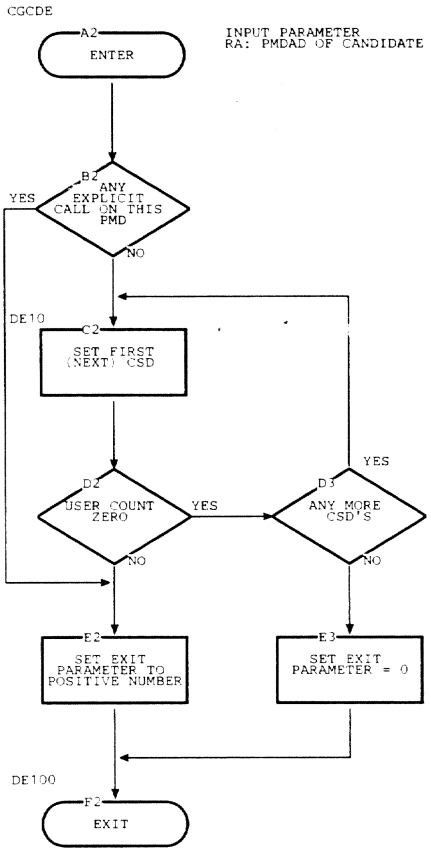
Program Logic Manual

GY28-2031-3

Dynamic Loader

Flowcharts on pages 91-144 were not scanned.

Chart BM. TEST USER COUNTS - CGCDE



APPENDIX A: ANALYSIS AIDS

Appendix A contains two symbol processing tables, a list of loader routines indexed by entry point name, a list of loader routines indexed by routine title, and a table of data references listed for each routine.

SYMBOL PROCESSING

Figures 18 and 19 summarize the symbol processing algorithms performed by the loader and unloader. Figure 18 shows the dynamic loader symbol lookup rules for resolving symbols in either explicit CALL/LOAD or DELETE adcon groups or in external REFs. Figure 19 shows the symbol posting rules for inserting DEFs into TDY hash chains.

The conditions set forth in Figure 18 are tested by the routine SET SEARCH FLAGS (CZCDLG), which produces as output a hash table pointer and library index (column 5).

The routine PCSA (CGCCT) checks the conditions in columns 1 and 2 of Figure 19 and performs the necessary CSECT attribute modifications indicated in column 3. The routine CHECK DEF LEGAL checks the conditions in columns 1, 2, and 4 to judge the admissibility of DEF symbols about to be posted. Finally, SELECT HASH (CGCCB) checks the conditions in columns 1, 2, and 4 before selecting the hash table pointer (column 5).

The symbols U, P, and O stand for normal user, system programmer, and operator or privileged system programmer, respectively.

1	2	3	4	5	
if	and	and	and	then	or
authority code is:	loader is resolving symbol from:	high-order bit of C1 or C3 byte of adcon group is:	control section containing adcon group or REF is:	lookup symbols in hash table:	If symbol is not found in hash table, search library:
U	Explicit LOAD/CALL or DELETE adcon group	0	SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			NONSYSTEM	USERHASH***	ALL**
		1	NONSYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			SYSTEM	USERHASH***	ALL**
	External REF	NA*	SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			NONSYSTEM	USERHASH***	ALL**
P or O	NA*	NA*	NA*	SYSHASHP or SYSHASHNP	ALL**

Notes:
*NA - not applicable, in the sense that the condition is not tested by the loader.
**ALL - the entire hierarchy of open libraries beginning at the last defined JOBLIB and ending with SYSLIB (or with that library yielding a valid definition).
***If the symbol to be resolved begins with SYS, the loader will look in SYSHASHP or SYSHASHNP, and then in SYSLIB.

Figure 18. Dynamic Loader Symbol Lookup Rules

1	2	3	4		5	6
if	and	then	and if		then	and
authority class is:	control section that contains DEF came from:	control section attributes may be altered:	control section that contains DEF has attributes:		DEFs may begin with only the symbols:	all legal symbols from control section are posted in:
U	SYSLIB	If control section is PRVLGD, loader sets SYSTEM attribute; hence, NON-SYSTEM and PRVLGD is impossible.	SYSTEM	PRVLGD	CZ, CHB	SYSHASHP
				NONPRVLGD	CZ, CHB	SYSHASHP
					All others	SYSHASHNP
			NONSYSTEM		Any but SYS	USERHASH
	USERLIB or JOBLIB	PRVLGD and SYSTEM erased.	NA*			
P	SYSLIB	PUBLIC and READONLY erased.	NA*		Any	SYSHASHP or SYSHASHNP
	USERLIB or JOBLIB	PUBLIC, READONLY, PRVLGD, SYSTEM erased.			Any but CZ, CHB	SYSHASHNP
O	NA*	PUBLIC and READONLY erased.	NA*		Any	SYSHASHP or SYSHASHNP
*NA - not applicable, in the sense that the condition is not tested by the loader.						

Figure 19. Symbol Posting Rules

DYNAMIC LOADER ROUTINE INDEX

These routines may be accessed from outside the loader:

Label (Entry Point)	Routine
CZCCD1	LOADER LOGOFF
CZCCD2	LOADER RELEASE
CZCCD4	LOADER CLEANUP
CZCDH1	LIB MAINT
CZCDL1	EXPLICIT LINK
CZCDL2	HASH SEARCH
CZCDL3	LIBE SEARCH
CZCDL4	PAGE RELOCATION
CZCDL5	MAP SEARCH
CZCDL6	SET SEARCH FLAGS
CZCDL7	Q-CHAIN
CZCDU1	EXPLICIT UNLINK
CZCDU2	DELETE MODULE

These routines are internal to the loader:

Label (Entry Point)	Routine
CGCCA	ALLOCATE MODULE
CGCCB	SELECT HASH
CGCCC	SRCHPACK
CGCCE	RESOLVE SYMBOL
CGCCH	LOAD PMD
CGCCJ	FIX PMD
CGCCK	ATTACH TEXT
CGCCL	FIX
CGCCN	ADD PMD
CGCCO	DROP PMD
CGCCP	REJECT DIAG
CGCCR	BISEARCH
CGCCT	PCSA
CGCCU	CHECK DEF LEGAL
CGCCV	LINK DEFS
CGCCW	GET STORAGE
CGCCY	DEFINE REF
CGCDA	MODIFY MUT COUNTS
CGCDB	DELETE CALLER MUTES
CGCDC	DELETE SELECTED MUTES
CGCDD	MODIFY USE COUNTS
CGCDE	TEST USER COUNTS
CGCDG	ADD MUTE
CGCDPR	LOADER PROMPT
CGCRQ	RESOLVE Q-REF
CGCSP	SETPAGE

For quick reference, both kinds of routines are listed alphabetically.

Routine	Label
ADD MUTE	CGCDG
ADD PMD	CGCCN
ALLOCATE MODULE	CGCCA
ATTACH TEXT	CGCCK
BISEARCH	CGCCR
CHECK DEF LEGAL	CGCCU
DEFINE REF	CGCCY
DELETE CALLER MUTES	CGCDB
*DELETE MODULE	CZCDU2
DELETE SELECTED MUTES	CGCDC
DROP PMD	CGCCO
*EXPLICIT LINK	CZCDL1
*EXPLICIT UNLINK	CZCDU1
FIX	CGCCL
FIX PMD	CGCCJ
GET STORAGE	CGCCW
*HASH SEARCH	CZCDL2
*LIB MAINT	CZCDH1
*LIBE SEARCH	CZCDL3
LINK DEFS	CGCCV
*LOADER CLEANUP	CZCCD4
*LOADER LOGOFF	CZCCD1
LOADER PROMPT	CGCDPR
*LOADER RELEASE	CZCCD2
LOAD PMD	CGCCH
*MAP SEARCH	CZCDL5
MODIFY MUT COUNTS	CGCDA
MODIFY USE COUNTS	CGCDD
*PAGE RELOCATION	CZCDL4
PCSA	CGCCT
*Q-CHAIN	CZCDL7
REJECT DIAG	CGCCP
RESOLVE Q-REF	CGCRQ
RESOLVE SYMBOL	CGCCE
SELECT HASH	CGCCB
*SET SEARCH FLAGS	CZCDL6
SETPAGE	CGCSP
SRCHPACK	CGCCC
TEST USER COUNTS	CGCDE

*Externally referable routines

DATA REFERENCES

Table 2 shows which tables are referenced by the loader's routines, by LOADER, LOGOFF, and by LIB MAINT. Entries are shown only for tables directly referenced by the routine itself; not for references by a called routine.

Table 2. Data References by Loader Routines

	TDY	MAP	ISA	DCB	SDST	TDT	MUT	VST
ADD MUTE	•						•	
ADD PMD	•		•					
ALLOCATE MODULE	•				•			•
ATTACH TEXT	•			•				•
BISEARCH	•	•						
CHECK DEF LEGAL	•		•					
DEFINE REF	•							
DELETE CALLER MUTES	•						•	
DELETE MODULE	•			•				•
DELETE SELECTED MUTES	•						•	
DROP PMD	•							
EXPLICIT LINK	•							
EXPLICIT UNLINK	•							
FIX	•							
FIX PMD	•							
GET STORAGE	•		•		•			•
HASH SEARCH	•		•					
LIBE SEARCH			•	•				
LINK DEFS	•							
Q-CHAIN	•							
LOADER PROMPT								
LOAD PMD	•			•				
MAP SEARCH	•	•	•					
MODIFY MUT COUNTS	•						•	
MODIFY USE COUNTS	•							
PAGE RE-LOCATION	•							•
PCSA	•		•			•		
REJECT DIAG	•							
RESOLVE SYMBOL	•							
SELECT HASH	•		•					
SET SEARCH FLAGS	•		•					
SRCHPACK								•
TEST USER COUNTS	•							
LOADER LOGOFF	•							
LOADER RELEASE	•		•			•		
LOADER CLEANUP			•	•		•		
LIB MAINT			•	•		•		
SETPAGE*				•				

*Also references PVT, RHD, MHD, DHD, and EPE

APPENDIX B: TABLES

Appendix B contains descriptions and examples of the following tables:

Task dictionary table (TDY).

PMD group
- PMD group header
- PMD preface
- PMD
  - CSD
    - Definition table
    - Reference table
    - Relocation dictionary
    - Virtual storage page table

Module usage table (MUT)

Memory map table (CHAMAP)

Hash tables (CHASHT and CHAUHT)

Vacant space table (VST)

ACCESS TO LOADER TABLES

Access to PMDs within PMD groups and to the PMD groups themselves is described in the discussion of the TDY. Access to the TDY is gained via the TDY heading; a pointer to this is set in the read-only half of the ISA. This pointer is identified symbolically in the ISA DSECT (CHAISA) as ISATDY.

Access to the hash tables and storage MAP table is through pointers in the TDY heading.

The module usage table (MUT) is maintained in the loader PSECT (CZCDLP) and is identified externally as CHBMUT. The location CHBMUT actually contains a pointer to the head of a chain of available blocks of MUTEs. The various MUT chains must be traced from their sources in the PMD prefaces (PAPA chain heads and BABY chain heads). If MUT space is exhausted in CZCDLP, additional space will be obtained by ADD MUTE (CGCDG).

TASK DICTIONARY TABLE (TDY)

The dictionary table (TDY) contains information needed to load (and unload) the modules in a particular task. It consists of a heading, three hash tables (two system and one user), the storage map table (MAP), and one program module dictionary (PMD) for each module loaded during the task (Figure 20). The PMDs are arranged in irregularly located PMD groups discussed in this appendix. TDY is initialized by STARTUP and maintained by the dynamic loader.

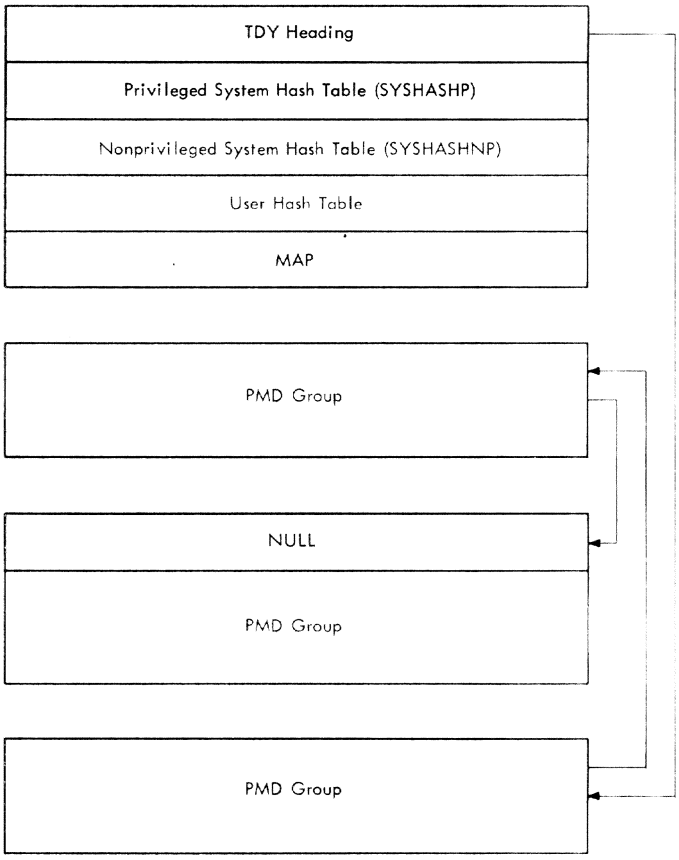


Figure 20. Task Dictionary Organization

TDY Heading (CHATDH)

The TDY heading (see Figure 21) is 16 words in length and contains:

Word 0 - Link to PMD group: The address of the first word of the last PMD group to be entered into the TDY.

Word 1 - Hash divisor: This is some number less than the length (in words) of each hash table. It is provided by STARTUP and remains unchanged during a task.

Word 2 - Pointer to system hasn table: The virtual storage address of the beginning of the privileged system hash table. The nonprivileged system hash table begins at

the end of the privileged system hash table.

Note: If the user authority is P or O, LOGON sets word 3 (pointer to user hash table) equal to the contents of word 2 (pointer to privileged system hash table).

Word 3 - Pointer to user hash table: The virtual storage address of the beginning of the user hash table.

Word 4 - The virtual storage address of the origin of MAP table.

Word 5 - The length, in bytes, of the maximum space allocated by the system for the task's storage MAP.

Word 6 - A count of currently valid MAP entries.

Words 7-15 - Several words reserved for future expansion.

0	Link to PMD Group
1	Hash Divisor
2	Pointer to Privileged System Hash Table
3	Pointer to User Hash Table
4	Pointer to MAP
5	Maximum Length of MAP
6	Length of Current MAP
7-15	Reserved

Figure 21. TDY Heading

PROGRAM MODULE DICTIONARY (PMD) GROUP

Each PMD group consists of at least one PMD with its associated PMD preface. When a new PMD is to be added to the TDY, if room exists in the same page that contains the last inserted PMD, the new PMD will be added to that page and become part of that PMD group. If such space does not exist in the page, a new PMD group is formed starting with the new PMD. Note that the first PMD in a group may exceed a page in length, but that successive PMDs in a group may not exceed a page. Space for PMD groups is allocated by the loader subroutine, ADD PMD (CGCCN).

PMD groups are chained together bi-directionally through the first two words in each PMD group header (see Figure 22). The TDY heading contains a pointer to the beginning of the chain of the PMD groups.

Pointer to Next PMD group header
Pointer to Previous PMD group header
Pointer to Last PMD in this group
Pointer to End of group

Figure 22. PMD Group Header

PMD Group Header

Each PMD group header consists of four pointers:

Word 0 - A pointer to the next PMD group header.
Word 1 - A back pointer to previous PMD group header.
Word 2 - A pointer to the last PMD in this group.
Word 3 - A pointer to the first byte past the end of this PMD group, which therefore defines the beginning of available space in this group.

The PMD group header is at the beginning of a page.

POINTER TO NEXT PMD GROUP HEADER: This either contains the address of the next PMD group header, or is zero if this is the last PMD group in the chain.

POINTER TO PREVIOUS PMD GROUP HEADER: This contains the address of word 0 of the previous PMD group header in the chain. The most recently added group will back-link to word 0 of the TDY heading (CHATDH).

POINTER TO LAST PMD IN THIS GROUP: This is the beginning of a circular chain of all PMDs in the group. It contains the address of the last (most recent) PMD preface in this PMD group. When the group collapses, this pointer will point to itself; the unloader routine DROP PMD (CGCCO) will recognize this condition and release the pages containing the collapsed group through FREEMAIN.

Figure 23 shows a sample PMD group.

POINTER TO END OF GROUP: Contains the address of the beginning of the available space at the end of the last page in the group. Space made available by deletion of a PMD within a group is not accounted for. Space is only released on a full group basis.

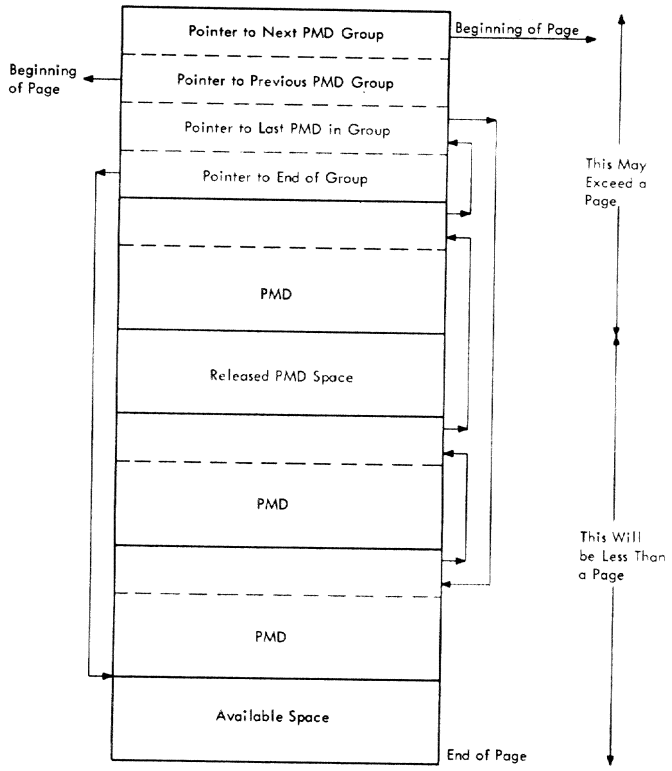


Figure 23. Sample PMD Group

PMD Preface

The PMD preface (Figure 24) is generated by STARTUP if the PMD preface is part of initial virtual storage or by the loader if it is not. The contents are maintained by the loader. It immediately precedes the PMD at load time; and when the term PMD is used in the following description, the PMD preface is generally inferred. The PMD preface contains the following entries:

Word 0 - A link to next PMD in the chain of PMDs within this PMD group. The PMD group header contains a pointer to the last PMD of this group. Since each new PMD is inserted at the beginning of the chain, PMDs are in reverse order of appearance within a PMD group. The link contains the address of the first word of the PMD preface of the next PMD. The last PMD in the chain points to the third word in the PMD group header.

Word 1 - A link to the MUTE (module usage table entry) chain for modules which explicitly call this module. This chain is further described under "Module Use Table." This entry is the beginning of the BABY MUTE chain for this module. It is zero if there are no entries in the chain; otherwise, it contains the address of the forward BABY link entry in the MUT chain.

Word 2 - A link to the MUTE chain for modules that are explicitly called by this module. This chain is further described under "Module Use Table." This entry is the beginning of the PAPA MUTE chain for this module. It is zero if there are no entries in the chain; otherwise, it contains the address of the respective forward PAPA link entry in the MUT chain.

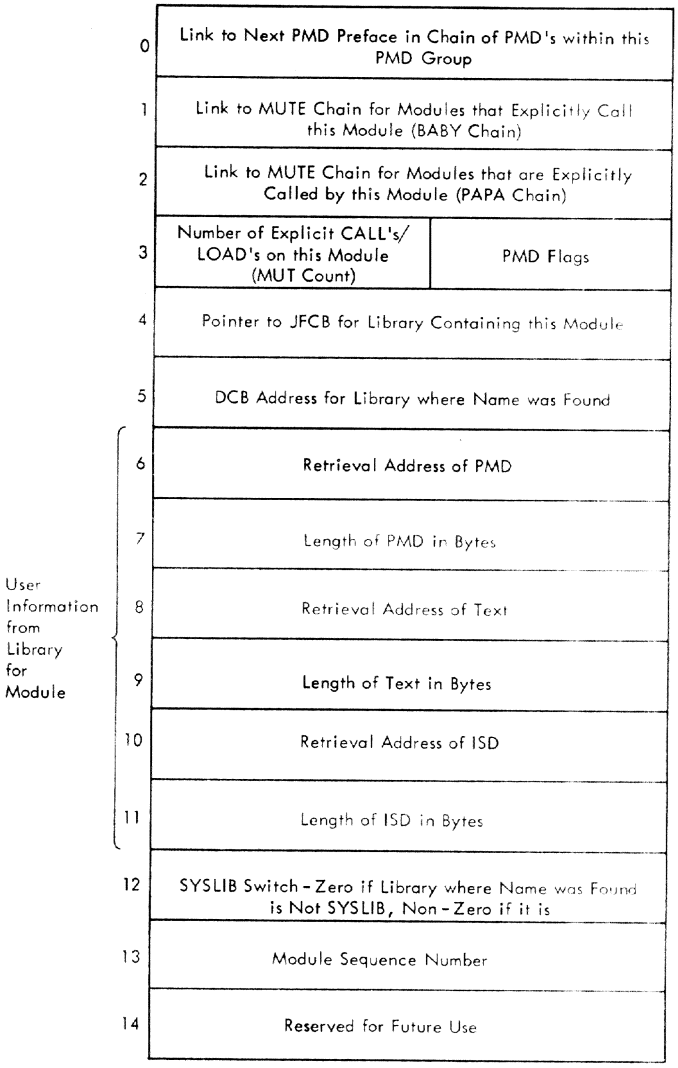


Figure 24. PMD Preface

Word 3 left half - The number of explicit links (CALLs/LOADs) to this module. For each explicit link to this module the value of this field is incremented by one. When the corresponding MUTE is removed

(see MUTE processing routines), the value is decremented. The contents are initially zero.

Word 3 right half - PMD Flags. The PMD flags field is a halfword containing flags used by the loader. The following flags are defined (bits numbered from left to right starting with 0):

Bit 15 - Public flag -- this bit is set if this module contains any public control sections.
Bit 14 - Condidate flag -- this bit is set if this module is on the deletion candidate list.

Word 4 - A pointer to the JFCB for library containing this module.

Word 5 - Address of DCB for library in which name resolved.

Words 6-11 - User information from library where this module was obtained.

The form of the retrieval address is:

Bits 0-15 - Relative page number.
Bits 16-31 - Zeros.

Note, therefore, that the retrieval address for the PMD will always be zero.

Word 12 - SYSLIB switch: set nonzero if module was extracted from SYSLIB.

Word 13 - The module sequence number. Each module is assigned a consecutive sequence number as it is loaded. This sequence number is used to differentiate unnamed (non-common) control section references among modules.

Word 14 - Reserved for future use.

PROGRAM MODULE DICTIONARY (PMD)

The output from an assembler, compiler, or the linkage editor is known as a program module. This is composed of a program module dictionary (PMD), text, and internal symbol dictionary (ISD).

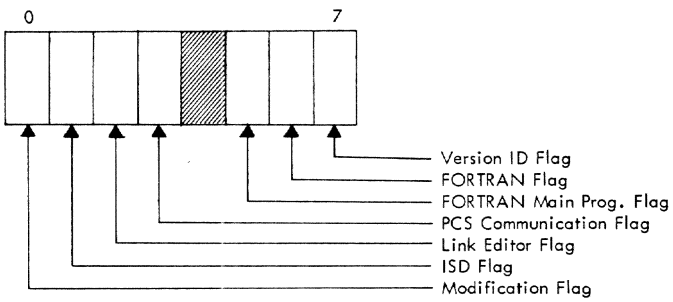
Each PMD consists of one PMD heading plus as many control section dictionaries (CSD) as there are control sections in the module. Address pointers in the PMD are initially relative to the beginning of the PMD itself (not the PMD preface), except where otherwise specified. Some fields in the PMD are filled in by the loader. These are left zero by the language processor. The PMD format is shown in Figure 25.

PMD Heading

1. Length of PMD in bytes: This length does not include the PMD preface.

2. Diagnostic code (1 byte): The diagnostic code indicates the highest level diagnostic encountered during generation of the module by the language processor that created it.

3. Flags (1 byte): The flag bits are numbered from left to right starting with zero and are defined as follows:



Bit 0 - In a system module indicates that module was modified by other than a language processor.
Bit 1 - This module has an associated ISD. This bit is set by the processor creating the PMD.
Bit 2 - This module was produced by the link editor.
Bit 3 - PCS is to be called before module is unlinked. This bit is set by PCS and examined by the unloader (CZCDU).
Bit 5 - FORTRAN flag, set by FORTRAN compiler.
Bit 6 - FORTRAN main program flag, set by FORTRAN compiler.
Bit 7 - Version ID indicator. If this bit is set, the module version ID 'TDYMID) is to be interpreted as a 64-bit binary number which is the creation date of the module expressed as the number of microseconds that have elapsed from March 1, 1900, until the time of module creation. If this bit is not set, the version ID is eight alphameric EBCDIC characters.

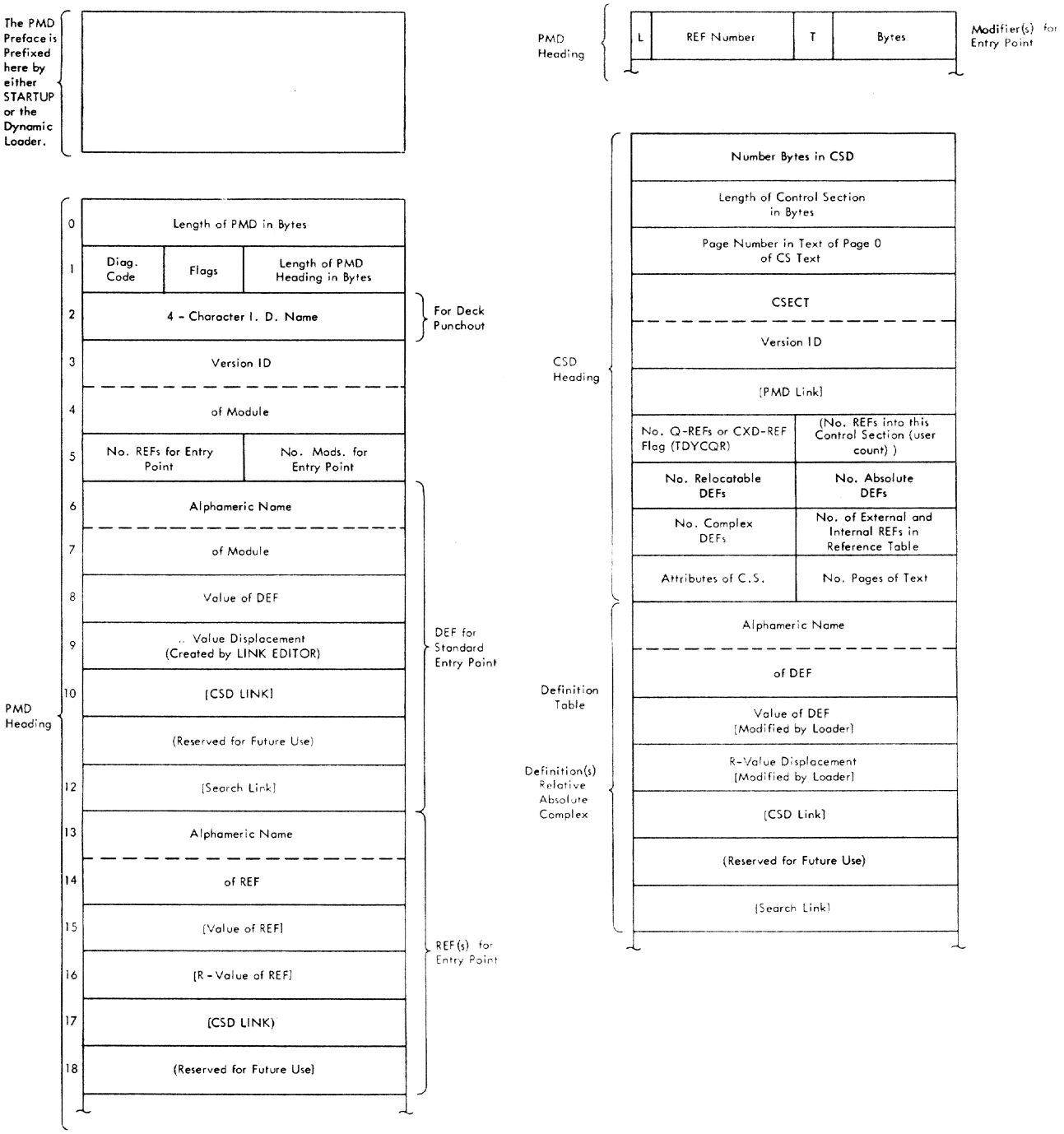


Figure 25. Format of PMD Entry (Part 1 of 2)

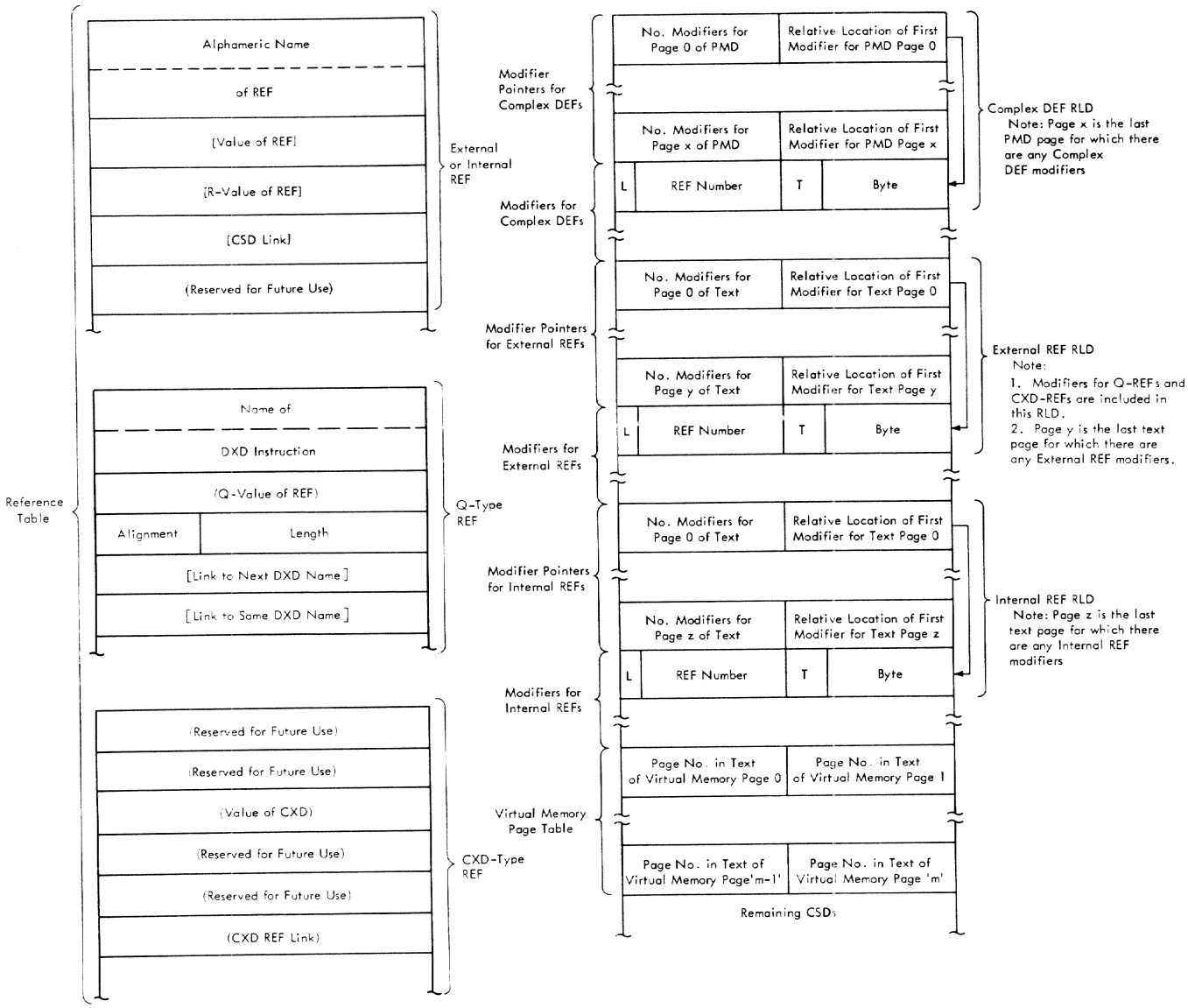


Figure 25. Format of PMD Entry (Part 2 of 2)

4. Length of PMD heading: This is the length of bytes of the PMD heading.

5. 4-Character I.D. Name: The 4-character I.D. name is supplied by the user to serve as deck identification if the module is punched into cards. This field is currently unused.

6. Version I.D.: See 3. above (bit 7 discussion) for interpretation of version I.D.

7. Number of REFs for the standard entry point: The DEF for the standard entry point is always treated as a complex DEF. This field contains the number of REFs. It may be zero.

8. Number of modifiers for the standard entry point: This field contains the number of modifiers that are to be used to compute the DEF for the standard entry point.

9. DEF for standard entry point: This seven-word entry describes the DEF for the standard entry point of the module. It has the same form as the individual DEF entries within the CSDs. The standard entry point DEF for the module belongs to the first PSECT of the module and is treated the same as a complex DEF whose ENTRY statement appears within that PSECT. If no PSECT is declared, the standard entry point will be associated with

the first CSECT instead. This DEF entry contains the following subfields, which are described in the discussion of DEF entries, under "Control Section Dictionary."

a. Alphameric name of module
b. Value of DEF
c. R-value displacement
d. CSD link
e. Reserved for future use
f. Search link

The alphameric name is also the name of the module.

10. REF(s) for entry point: These have the same form and function as the REFs described in the CSD discussion, following.

11. Modifier(s) for standard entry point: These have the same form and function as the modifiers described in the CSD discussion below except that they apply to standard entry point DEF.

CONTROL SECTION DICTIONARY (CSD)

The control section dictionary is made up of:

1. CSD heading
2. Definition table
3. Reference table
4. Relocation dictionaries (RLDs)
5. Virtual storage page table (VMPT)

CSD Heading

1. Number of bytes in CSD: This field specifies the length of the control section dictionary in bytes.

2. Length of control section in bytes: This specifies the virtual storage span of the control section. The length of the virtual storage page table is derived from this length. For example, if the length of the control section is 8192, the virtual storage page table will contain two entries, but if the length is 8193 bytes, the virtual storage page table will contain three entries. This value will be equal to the highest location counter value assigned by the language processor, plus one.

3. Page number in text of page 0 of control section text: The text for each control section in the module occupies an integral number of pages in its resident data set. The text pages for all control sections in a module are contiguous. This number is the page number, relative to the first page of text for this module, of the first page of text for this control section. (Numbering begins with 0.)

4. Version I.D.: This is a 64-bit binary number which is the creation date of the control section expressed as the number of microseconds that have elapsed from March 1, 1900, until the time of control section creation.

5. PMD Link: The PMD link is filled in by STARTUP or the dynamic loader. It points to the beginning of the PMD preface.

6. Number of Q-REFs or CXD-REF flag (TDYCQR): The number of Q-REFs in this control section, and whether there is a CXD-REF in this control section.

Bit 0 (leftmost): Set to 1 if there is a CXD-REF.
Bit 1: Not used.
Bits 2-14: The number of Q-REFs.

7. Number of implicit references to this control section (user count): This is a count of the number of REF entries that refer to this control section and are linked to this CSD through their CSD link. It is computed by the loader. It includes both external and internal references. This number is arbitrarily set to X'7FFF' by STARTUP for each control section in initial virtual storage (IVM) to prevent unloading of IVM modules.

8. Number of relocatable definitions: This is the number of relocatable definitions in the definition table. It is always at least one; namely, the control section name DEF.

9. Number of absolute definitions: This is the number of absolute definitions in the definition table. It may be zero.

10. Number of complex definitions: This is the number of complex definitions in the definition table. It may be zero.

11. Number of references from this CSD: This is the sum of external and internal references in the reference table. It may be zero.

12. Attributes: This halfword has one bit set for each attribute possessed by the control section. Currently defined attributes are shown below.

Bits are numbered from left to right, starting with 0.

a. Public Name (Bit 0 on)
This is used only by the dynamic loader to specify nonblank control sections whose names appear in the shared data set table (SDST). The first such control will appear in the SDST under the module name. A control section may be indicated as both having a public name and rejected.

b. CSD has been allocated storage (Bit 1 on)
Set by the dynamic loader, if applies.

c. PCSA (CGCCT) called for this CSD (Bit 2 on)
Set by the dynamic loader, if applies.

d. Public Storage Assigned by CONNECT (CZCGA7) (Bit 3 on)
Set by the dynamic loader, if applies.

e. Bits 5 and 4 are not used.

f. Common CSECT Rejected (Bit 6 on)
The dynamic loader sets this flag to indicate to the Program Control System that the CSECT was rejected as a common CSECT that was already loaded in another module.

g. TDYCQR Validity (Bit 7)
The dynamic loader sets this flag to indicate that the count of Q-REFs in TDYCQR is valid. If bit 7 is off, the count of Q-REFs is not valid.

h. System (Bit 8 on)
Any external symbol that appears in a control section with the system attribute can not be referenced by a user program unless the symbol begins with SYS. Conversely, no reference from a control section with a system attribute may be to a user symbol.

i. Privileged (Bit 9 on)
A control section with a privileged attribute is assigned storage key C, which provides fetch as well as store protect. This attribute overrides R/O. Anything in a privileged CSECT may be referenced only when the PSW key is zero.

j. Common (Bit 10)
A common section is a control section common to all modules in which it is declared. Common sections are more fully discussed in Linkage Editor and Assembler Language.

Common sections are of two types:

(1) Named common sections (those with a name not all blanks). These are treated as fixed-length sections.

(2) Blank common sections, whose name consists of eight blanks. FORTRAN blank common is assigned the variable and common attributes by the FORTRAN compiler.

The treatment of blank common sections differs from that of blank non-common sections. Control section rejection is instituted between blank common sections of different modules, whereas blank noncommon sections of different modules are treated as independent control sections. The latter are called unnamed control sections.

k. PSECT (Bit 11 on)
If this bit is set, the dynamic loader overrides the system packing indicator and inserts this control section as packed.

l. Public (Bit 12 on)
Control sections are not shared by control section name alone. A public control section of a module residing in a given data set (library) is shared if another user has access to the same data set and module. Control sections of a given module need not all be public or nonpublic. Fixed-length public control sections with the same attributes are assigned storage in the same assignment. A public control section must never contain relocatable adcons (A-,V-, or R-type).

m. Read-only (Bit 13 on)
Read-only specifies that the control section may not be stored into. It causes memory protection by means of a storage class B assignment to all pages of the control section. Non-read-only and nonprivileged control sections are assigned storage class A.

n. Variable-length (Bit 14 on)
A variable-length control section will be allocated pages in excess of the length stated in the CSD heading.

o. Fixed-length (Bit 14 off)
A fixed-length control section will be allocated a fixed number of pages at load time.

13. Number of pages of text: This specifies the number of pages of text for this control section in the data set. It should be noted that this generally does not correspond to the number of pages in the virtual memory page table. It cannot, of course, be larger.

Definition Table

The definition table is made up of seven-word entries, one for each external definition in the current control section. Definitions are grouped as relocatable, absolute, and complex in that order. The first definition in the table is the name of the current control section.

A relocatable definition is an external definition whose value may be computed as the sum of the origin of the control section wherein it appears and a constant which is the symbol's displacement from the section origin.

An absolute definition is an EQU item with an absolute value whose name has been declared an entry point in the control section in which the name is defined.

A complex definition is either an EQU item with a complex relocatable value (that is, containing external symbols) or a simple relocatable definition whose ENTRY statement appeared within a control section other than the section in which it is defined. The definition entry appears within the CSD of the control section that contains the ENTRY statement. (Note that the origin of the same control section is the R-value for the DEF.) The complex DEF is required in this case, with one REF entry that names the control section in which the DEF symbol is actually defined.

Each DEF in the definition table contains entries of the following form:

1. Alphameric name of DEF: This field contains the eight-character alphameric name of the DEF.

2. Value of DEF: The value of the DEF is set by the language processor and is modified by STARTUP or the loader in the case of complex and relocatable definitions. For relocatable DEFs, the value portion of the definition entry contains the displacement value of the symbol relative to the base of its control section. For absolute DEFs, this entry contains the absolute value; for complex DEFs it contains the absolute portion of the DEF value, which may zero.

3. R-value Displacement: The displacement for R-value word contains the displacement of the original defining control section origin with respect to the head of the control section within which the definition now appears. This is required to compute valid R-values for control sections that have been combined by linkage editing. In creating the PMD, only the linkage editor will ever produce a nonzero value in this word.

4. CSD link: The CSD link is initially zero. It is filled in by STARTUP or the dynamic loader when the control section is loaded as a pointer to the beginning of the CSD in which this DEF appears, provided neither the DEF nor the control section has been rejected.

5. For future use.

6. Search link: This field is filled by the HASH SEARCH routine of either the loader or STARTUP. It contains the address of the beginning of the next DEF entry which hashes to the same value. It contains zero if there are no more DEFs with the same hash value in this chain.

Reference Table

The reference table is made up of six-word entries, one for each external symbol referenced within the control section. Each entry for an external or internal REF contains:

1. Alphameric Name of REF: This field contains the eight-character alphameric name of the REF.

2. Value of REF: This is filled in by STARTUP or the dynamic loader. It contains the value of the DEF to which the REF refers. If the DEF is undefined, it contains the address of a portion of virtual storage wherein reference is illegal.

3. R-value of REF: This is filled in by STARTUP or the dynamic loader. It contains the virtual storage address of the beginning of the control section in which the DEF appears. This value is obtained from the R-value displacement word of the satisfying DEF entry. If the DEF is undefined, this word contains the address of a portion of virtual storage wherein reference is illegal.

4. CSD Link: This pointer, initially zero, is filled by STARTUP or the

dynamic loader. It points to the beginning of the CSD in which the DEF that defines this REF appears. If a corresponding DEF could not be found upon the appearance of a REF, the CSD link is to the beginning of the CSD wherein the REF itself appears.

5. For future use.

Each entry for a Q-REF contains:

1. Name of DXD instruction: The eight-character alphameric name of a DXD instruction.

2. Q-value of REF: This is filled in by the RESOLVE Q-REF routine of the dynamic loader. It contains the displacement from the beginning of the combined dummy sections of the dummy section defined by the DXD instruction.

3. Alignment, Length: The alignment and length specified by the assembler language processor.

4. Link to Next DXD Name: This is filled in by the Q-CHAIN routine of the dynamic loader when Q-CHAIN posts the REF on one of the eleven hash chains for Q-REFs.

5. Link to Same DXD Name: This is filled in by the Q-CHAIN routine of the dynamic loader when Q-CHAIN posts the REF on one of the secondary Q-type REF chains for duplicate-name DXDs.

Each entry for a CXD-REF contains:

1. For future use.

2. Value of CXD: This is filled in by the EXPLICIT LINK routine of the dynamic loader. It contains the length of the combined dummy sections.

3. For future use.

4. CXD REF Link: This is filled in by the ALLOCATE MODULE routine of the dynamic loader as CXD-REFs are chained together.

Relocation Dictionary (RLD)

Three RLDs appear in each control section dictionary. They are:

1. RLD for complex definitions

2. RLD for internal references

3. RLD for external references

Each RLD has the same format, consisting of modifier pointers and modifiers. The RLD for complex definitions differs in that pages mentioned in this table are pages of the PMD rather than the text.

Modifier Pointer: Modifier pointers are used to designate the application of modifiers to adcons on appropriate pages of text (or of the PMD for complex DEFs). The first modifier pointer applies to the first text page, the second modifier pointer to the second text page, etc. Null (textless) pages do not have modifier pointers. There always exists at least one modifier pointer for an RLD. However, there need not be a modifier pointer for each page of text; the modifier pointers may be ended at the last text page for which there exists any modifier.

The modifier pointers consist of two fields, in the left and right halfwords:

Left half - Number of modifiers for page

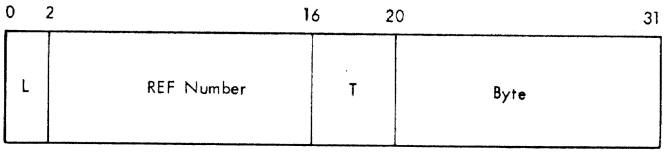
This field contains the number of modifiers that apply in this page.

Right half - Location of first modifier for this page

This contains the locations, in bytes, relative to the right half of the pointer itself for the first modifier for this page. If there are none, it points to the location where one would have appeared if there were any.

A special note should be made of the technique for determining the length of an RLD. The location of the first modifier for this page is in the right half of the first pointer for the RLD. In the word preceding the first modifier word is the last modifier pointer for the RLD. Adding the location of the right half to the contents of the right half of the last pointer gives the beginning of the last set of modifiers. Add to this four times the number of modifiers in the last set to get the end of the RLD.

Modifier: The modifiers themselves are each a fullword and are divided into 4 fields:



1. L: L (2 bits) is the length, in bytes, of the adcon to be modified. A value of zero indicates a fullword (4 bytes).

2. Ref Number: Reference number (14 bits) is the ordinal number in this CSD's reference table of the reference whose definition value is to be used in modifying the adcon. References are numbered starting with zero.

3. T: T (4 bits) is the operation to be performed in modifying the adcon by the definition value. The values of T currently defined are as follows:

   a. Addition (T = 1)

      The definition value is added to the field of L bytes at the location specified by "Byte."

   b. Subtraction (T = 2)

      Same as addition, except that the definition value is subtracted from the field of L bytes.

   c. R-value (T = 3)

      The R-value of the REF is stored into the field of length L at the location specified by "Byte."

   d. Q-value (T = 4)

      The Q-value of the REF is stored into the field of length L at the location specified by "Byte."

   e. Value of CXD (T = 5)

      The value of the CXD instruction is stored into the field of length L at the location specified by "Byte."

4. Byte: Byte (12 bits) is the displacement in bytes (from the origin of its original containing page) of the adcon to be modified. Note that since PMDs are packed to word boundaries, this displacement will be added to an address for complex DEFs which generally is not a page boundary.

RLD for complex definitions: The format of these modifiers is as described above. These modifiers apply to the values of complex definitions; that is, the byte addresses in the modifier will be to the value words of complex DEF entries in the definition table, and the page numbers in the modifier pointers are for pages of the program module dictionary itself.

RLD for text external reference: This relocation dictionary is in the same form as described above. It has one pointer for each page of program text up to that text page that is the last to contain an adcon, and appropriate modifiers for each adcon in the text that refers to a symbol defined externally to this module. The page numbers are based on the first page for this control section, beginning with 0.

RLD for text internal reference: This is identical to RLD for text external reference above, except that the modifiers apply to adcons in the text that reference symbols defined within this module, such as control section names. This permits communication between control sections of the same module that may be allocated noncontiguous virtual storage.

Virtual Storage Page Table (VMPT)

This table has a halfword for each page of virtual storage that the control section occupies, beginning with page 0 and continuing upward in order.

The contents of each entry will be either:

1. All bits if the corresponding page is empty as a result of a DS or ORG statement.

2. The number of the page in the text relative to the beginning of text for this control section if the page contains code or data. This value multiplied by four becomes an index into both the external and internal RLDs, and is used to select the correct modified pointer word for adcon relocation.

This table is the means by which the text of the control section is related to the virtual storage assigned the control section. This is necessary because language processors do not necessarily output a byte of text for each byte of virtual storage assigned; that is, large ORG and DS statements may result in pages of text being skipped.

If for example, a source program were to begin with

```
ORG     10000
```

there would be no text output for the first two pages of virtual storage and the first page of text would correspond to the third page of the user's virtual storage. The first two VMPT entries would be all bits, and the third would contain zero. Within a page, however, the bytes of text correspond directly to the bytes of virtual storage. Thus, in the example above, the first page of text would represent virtual storage locations 8192-12287, and the first 1808 bytes of the page of text would be vacant (10000 - 8192=1808). The pages of text will always begin on page boundaries within the text module.

MODULE USAGE TABLE (CHAMUT)

Purpose

For each CALL (in this description, "CALL" will be used when referring to explicit CALL or LOAD) during a task, a MUT entry (MUTE) is created by the explicit linkage routine, ADD MUTE (CGCDG). It provides a record of all CALLs on and by modules in the task and is used to ensure correct unlinkage of called modules. See Figure 26.

Links and Addresses

All addresses and links used in the MUT are 32-bit virtual storage addresses.

Location of MUT: Each task creates and uses its own MUT. The head of each MUT and space for a number of MUTEs reside in the loader PSECT (CZCDLP) for that task. If additional MUT space is required, the MUT is extended by allocating a new page via GETMAIN.

Location of MUTEs: Prior to use, the MUT is initialized so that a chain of available space entries is anchored by a pointer in the loader PSECT. This cell is labeled externally as CHBMUT. A newly created MUTE is assigned space in the first available space, and the available space chain is relinked. When a MUTE is deleted, the released space is added to the beginning of the available space chain.

Contents of MUTE

A MUT entry is linked into two chains which have their origin in two different PMDs. The MUTE serves to tie together a called module and its explicit calling module. When a MUTE is created, it is linked bi-directionally into the calling PMD's

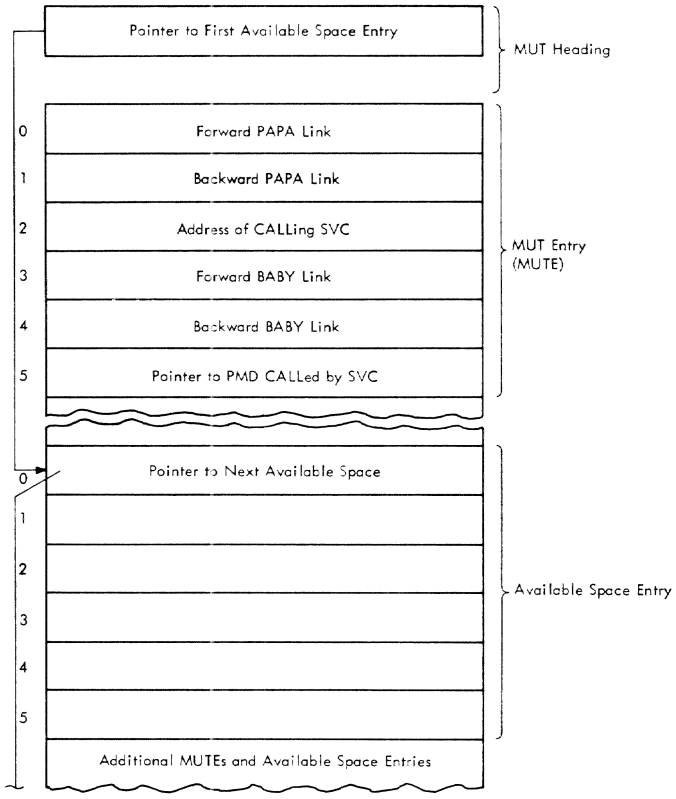


Figure 26. Format of MUT, MUTE Entry, and Available Space Entry

PAPA chain through the PAPA link words in the MUTE. The MUTE is also linked bi-directionally into the called PMD's BABY chain through the BABY link words in the MUTE. Thus if A calls B and C, two MUTEs and three chains are created (see Figure 27):

1. A PAPA chain that originates in A and links to both MUTEs.
2. A BABY chain that originates in B and links to one MUTE.
3. A BABY chain that originates in C and links to the other MUTE.

In addition to the links, each MUTE contains:

1. The address of the SVC that initiated the CALL. This information is used in explicit unlinkage to rearm the SVC.
2. The address of the PMD named by the CALL. This address is required during explicit unlinkage.

Adding MUTEs

When an entry is added to the MUT, its chains are linked, as follows:

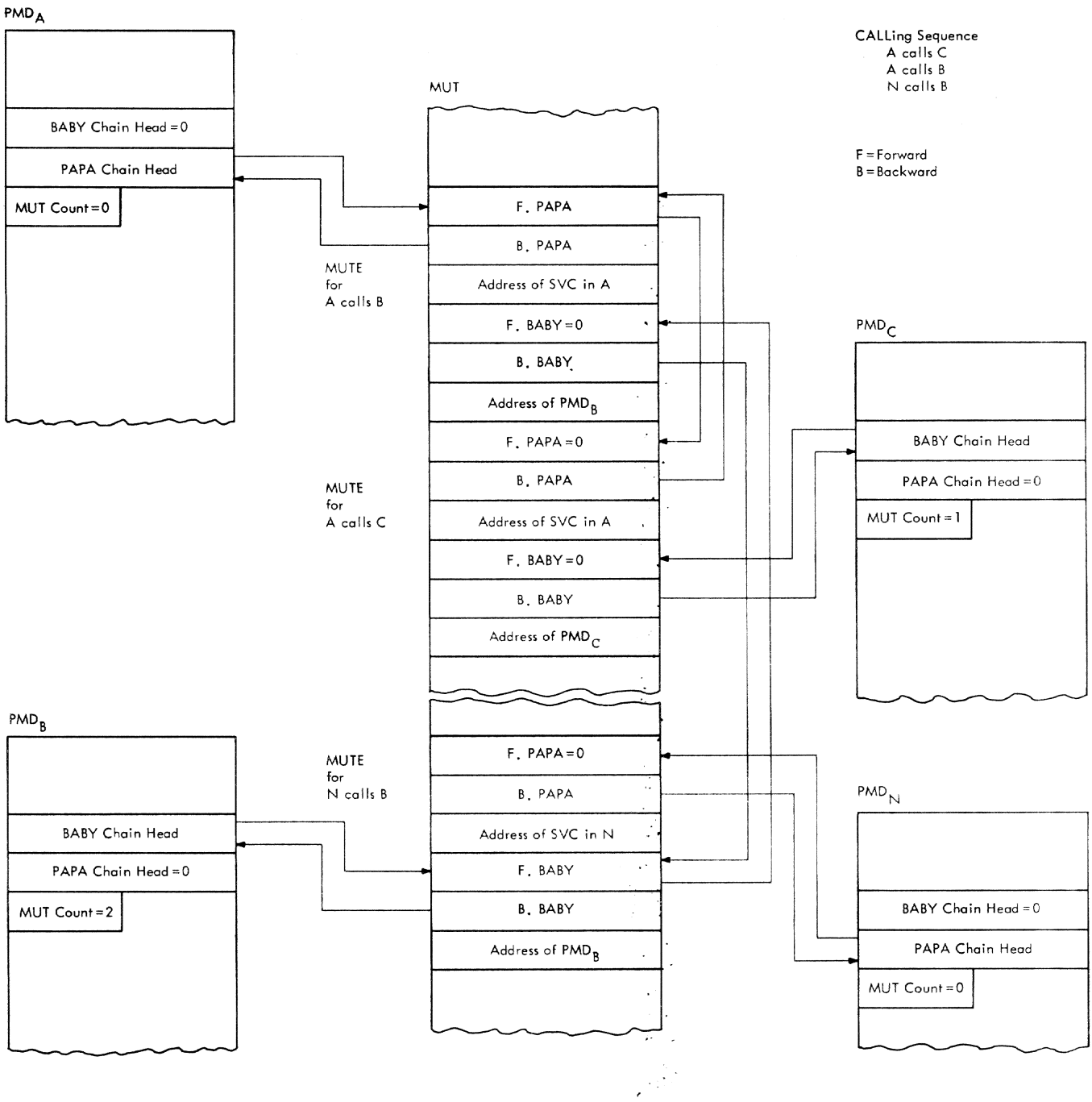


Figure 27. Diagram of Sample MUT, Showing Linkages and Appropriate PMDs

The new MUTE becomes the first MUTE in both its BABY and PAPA chains. The heads of the respective chains within the chain origin PMD will point to the new MUTE's forward BABY and PAPA links. The new MUTE's forward links themselves will point to the forward links of the previously entered MUTEs. If there is no previous entry, the forward link(s) will contain 0.

The backward BABY and PAPA links of each MUTE point to the respective forward BABY and PAPA links of subsequently entered MUTEs. Thus, the backward BABY and PAPA links of the last MUTE to be entered will point ot the BABY and PAPA chain heads in the respective called and calling PMDs.

Deleting MUTEs

When a module is to be deleted (see "Explicit Unlinking," Section 4), all MUTEs for CALLs it made, as well as for CALLs made upon it, are deleted. In other words,

all entries in both its BABY and PAPA chains are removed from the MUT. Such deleted MUTEs are added to the chain of available MUTEs that are threaded through the forward PAPA link words.

MUT Count

Each time a MUTE is created, the MUT count in the called PMD is incremented by one. When a MUTE is deleted, the corresponding MUT count is decremented by one. This count is used during the unlinking process to determine whether or not a module may be deleted; that is, modules whose MUT counts are nonzero at a certain point in the unlinking process are judged to be ineligible for deletion because of the existence of outstanding explicit references.

STORAGE MAP TABLE (CHAMAP)

The loader maintains a storage map table (MAP) which contains one entry for each control section involved in the allocation. (See Figure 28.) The table is a contiguous, ordered set of two-word entries. The first word contains the virtual storage address of the beginning of the control section; the second word contains the virtual storage address of the beginning of the corresponding CSD in the TDY. The table is maintained in compact ascending order according to the 32-bit, unsigned, virtual storage address of the beginning of the control section.

The maximum size of the table is established at STARTUP time. A pointer to the beginning of the table, the maximum length of the table in bytes, and the current length of the table are specified in the TDY headings.

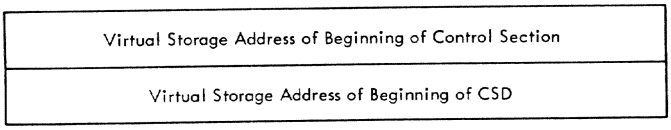


Figure 28. Memory MAP Entry

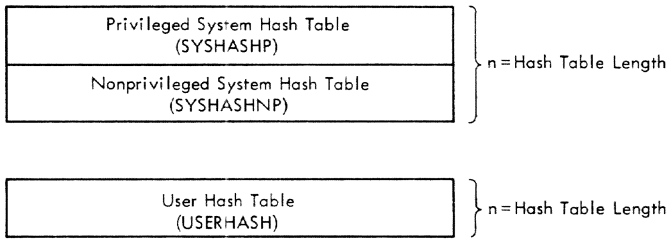
HASH TABLES (CHASHT AND CHAUHT)

There are three hash tables whose location and length are specified in the TDY. The system hash tables contain hash entries for SYSxxxxx symbols and symbols which appear in control sections with the system attribute. The user hash table contains all other symbols.

A pointer in the TDY points to the origin of the privileged system hash table. Nonprivileged system symbols (those not beginning with CZ or CHB) are contained in the nonprivileged system hash table, which immediately follows the privileged table.

Each hash entry consists of a single word which either contains the address of the beginning of the first corresponding DEF chain or is zero. The position of a symbol in the hash table is determined by the hashing algorithm. Whenever DEFs with different names hash to the same hash value, they are linked together through their search links.

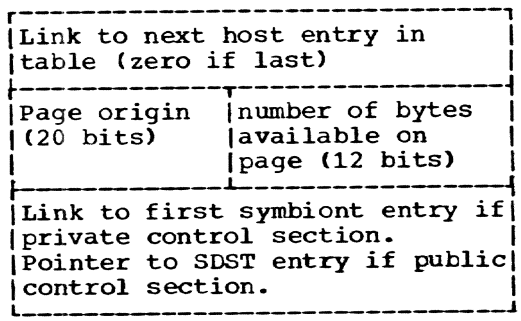
The hash value, H, is obtained by an exclusive OR of the first four characters of name with the last four characters. This resultant 32 bits is now divided by the hash divisor, and the remainder of this division multiplied by 4 is a relative index into the appropriate hash table.



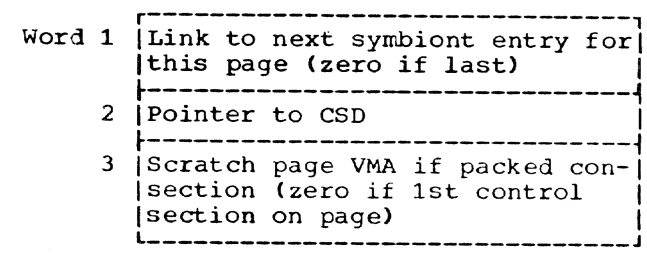
VACANT SPACE TABLE (VST)

The vacant space table is used for control section packing and contains three types of entries (host, symbiont, and available space entry) each three words in length.

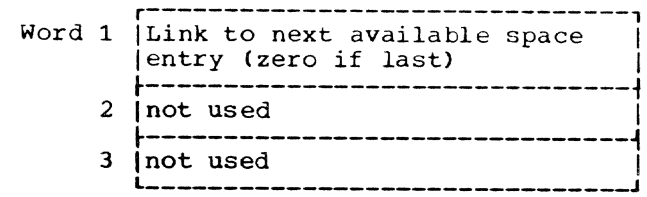
Host Entry



Symbiont Entry



Available space entry



A host entry is created for the first control section in a control section group at a page boundary. It reflects unused space on the last page of the control section group (if the last page contains text). Host entries are maintained in ascending order of the number of bytes available on the page. Host entries are not created for public pages if the amount of unused space is less than 8 bytes. Host entries for private pages are maintained until the relocation of the page is requested. If a host exists for the page being relocated, it is deleted and relinked into the available space entry list.

Symbiont entries are created for all but the first control section in a private control section group. Hence the first control section (host) owns the page, and secondary control sections (symbionts) are packed into the page.

Symbiont entries are not created for public control sections. A host entry for a public page points to the SDST entry (host) that owns the page. If a control section group is packed into a public page, the SDST entry for the group is flagged as symbiont.

A vacant space table pointer for each storage class and an available space pointer are maintained in the loader PSECT. These are initially zero. The first time SRCHPACK is requested to create a table entry, a page of virtual memory is obtained and available space entries are threaded to the available space pointer.

Table pointers in the loader PSECT are as follows:

Adcon	Address constant
CHAISA	DSECT for ISA
CHARCS	DSECT for recursive storage block (in Resolve Symbol)
CHASDM	DSECT for SDM (shared data set table entry)
CHASHT	DSECT for SYSHASH
CHATDH	DSECT for TDY heading
CHAUHT	DSECT for user hash table
CHBMUT	Entry point name of module usage table (in Unloader PSECT)
CSD	Control section dictionary
CSECT	A control section other than a prototype (PSECT) or COMMON or DSECT
DCB	Data control block
DDEF	DDEF command or macro instruction
DEF	External symbol definition
EPE	External page entry
GR	General register
ISA	Interrupt storage area
ISATDY	Pointer in ISA to TDY
ISAUTH	User authority cell in ISA
ISD	Internal symbol dictionary
IVM	Initial virtual storage
JFCB	Job file control block
JOBLIB	A user library created by a DDEF command with a JOBLIB keyword
MAP	Memory map table
MUT	Module usage table (in Unloader PSECT)
MUTE	Module usage table entry
O	Authority code for the system operator or privileged system programmer
P	Authority code for the system programmer
PCS	Program control system
PMD	Program module dictionary
PSECT	A prototype control section
PVAOT	Pseudo vector available offset table
REF	External symbol reference
RESTBL	Relative external storage correspondence table
RLD	Relocation dictionary
RN	REF number
SDST	The shared data set table
SPT	Shared page table
SYSHASHNP	The nonprivileged system hash table
SYSHASHP	The privileged system hash table
SYSLIB	The system library
TDT	Task data definition table
TDY	Task dictionary
U	Authority code for a "normal user"
USERHASH	The user hash table
USERLIB	The user library
VMA	Virtual storage address
VMPT	Virtual storage page table
VST	Vacant space table
XPT	External page table

APPENDIX D: LOADER RESTRICTIONS

The following restrictions are imposed on the TSS/360 user. Those marked with an asterisk are enforced by the dynamic loader:

1.* The user is not allowed to declare either system or privileged control sections. The dynamic loader will erase such attributes from control sections that are not contained in modules extracted from SYSLIB. This has a secondary effect: the user is prevented from declaring entry points whose names begin with the letters SYS. The user may declare any other form of entry point name.

2. No user-written program called by type-III linkage nor any routine called by such a program may execute a LOAD, explicit CALL, or DELETE statement.

3.* Complex DEFs may not be defined in terms of another complex DEF within the same module. Such a situation could result in a complex definition loop, which the loader protects against by not allowing such symbol resolution in any case. Note that this situation can only arise through control section rejection or through the link-editing together of two modules. For example:

```
Module A
P1    PSECT
      ENTRY   E1
C1    CSECT
E1    EQU     *
      END

Module B
P2    PSECT
      ENTRY   E2
E1    CSECT
E2    EQU     *
      END
```

If A and B are link-edited together, and CSECT E1 is deleted by the prior occurrence of DEF E1 in module A, complex DEF E2 will be defined in terms of complex DEF E1 within the same module, a situation which the loader will not allow.

If A and B are not link-edited, however, and the loader is called to LOAD first A and then B, CSECT E1 will be rejected by complex DEF E1 in A. This will result in complex DEF E2 being defined in terms of complex DEF E1 in a different module. The loader will allow this situation, but will issue warning diagnostic.

4. The user is advised that executing a CALL or LOAD of a CSECT name will return an R-value equal to its V-value; namely, the origin of the CSECT. Thus, calling reenterable routines by CSECT name will result in the PSECT address not being available when the routine is entered.

5. The loader will issue diagnostics for undefined external references and substitute for them an illegal address which will usually cause an address specification error at attempted use by the task. Such an error may not be produced, however, if the user should dynamically modify a V-con containing an undefined REF.

6.* The user is advised to include no adcons within public CSECTs. The loader will refuse to resolve adcons appearing on public pages.

7.* Control section rejection arises from two sources: If a control section name is illegal, the control section is rejected by the loader, or if a control section name duplicates the name of some previously entered DEF (not necessarily a control section name), the loader will reject the control section. Control section rejection means that none of the DEFs in the rejected CSD are to be included in the allocation, and that any REFs that might have been satisfied by such DEFs must either be satisfied elsewhere or go undefined. Control section rejection may be caused by some anomalous circumstance which will result in a diagnostic advising the user. Such diagnostics and their causes are listed under "Loader Diagnostics."

8.* No module may delete itself by a DELETE operation, either as a primary or secondary deletion candidate. The loader protects against this by removing the deleting module from the candidate list if it is ever posted there.

9.* The primary deletion candidate may not be deleted if there are any outstanding implicit references to it after the candidate list is construc

10.* Secondary deletion candidates may not be deleted if there are any outstanding implicit or explicit references to them after the candidate list is constructed.

11. The loader will allow a module that is loaded or called by another module to delete that module, using a DELETE procedure. The user is therefore advised not to attempt to return to the "caller" in such a case, as it will have disappeared. For example, module A calls module B, whose first step is to DELETE A. Any attempt by B to return to A may cause an addressing exception error and is in any case a programming practice that is to be avoided.

12. The following discusses TSS/360's treatment of unnamed control sections. Unnamed control sections arise from three sources: a CSECT statement with a blank symbol field, omission of any CSECT statement, and declaring of blank COMMON. Blank COMMON carries a name of eight alphameric blanks; all others carry a name of sixteen hexadecimal zeros. Unnamed noncommon control sections are treated as unique to the module in which they were declared by the following devices:

    a. When the linkage editor processes unnamed CSECTs, it substitutes a unique number in the lower eight hexadecimal digits of the name of each such CSECT it processes. In addition, the name part of each REF (in the original module) that references the unnamed CSECT is also modified to match the altered CSECT name. This technique preserves unnamed references in the event two such CSECTs are combined during link editing.

    b.* When the dynamic loader processes unnamed CSECTs (identified by the fact that the first half of the CSECT name consists of eight hexadecimal zeros), it adds further modification to the name. Each module as it is loaded is assigned a unique number, a module sequence number, by the loader. The loader places this number in the lower four hexadecimal digits of the first half of the name of each unnamed noncommon CSECT and each REF in the module that references the CSECT. The combination of the linkage editor's and the dynamic loader's treatment of unnamed noncommon CSECTs will prevent CSECT rejection among such CSECTs. Of course, blank COMMON sections may cause rejection; the blank name is handled just as any other name. This is reasonable, since the concept of control section rejection was instituted for the purpose of tying together common references across modules.

13.* Privileged system service routines may define only external symbols that begin with the letters CZ or CHB. This means that all normal entry points, CSECT names, and module names must conform to this naming standard. Furthermore, nonprivileged system routines may not define external symbols beginning with CZ or CHB. (These restrictions do not apply to user programs.)

APPENDIX E: DIAGNOSTIC MESSAGES

This section contains a list of diagnostic messages that may be issued by the dynamic loader. Each diagnostic contains an explanation of the cause of the diagnostic, with loader action, where applicable. In this list, brackets are used to denote that a substitution will be effected by the loader; that is, that such bracketed portions of the message are variable. Those diagnostics that result in additional serious load error indication by the loader are marked by an asterisk (*) after the diagnostic number.

DIAGNOSTIC	COMMENT
1. CZCDL002 CANCELLED: SYMBOL [x] TO BE {CALLED / LOADED} NOT FOUND	The loader is unable to find the symbol, x, that occurred in the operand field of an assembly LOAD or CALL statement or a command language LOAD or RUN statement or, if it was found, all control sections were rejected in attempting to load it. It may also be true that a module was in fact loaded that defines x, but that x was resolved from a system module which the user is unable to reference, since x is posted in the SYSHASHP or SYSHASHNP table.
2. CZCDL011 CANCELLED: LIBRARY SEARCH ERROR FOR [x]	The loader used the FIND function of VPAM to locate symbols in the external libraries. If FIND is called to locate some symbol, x, and returns with a code indicating some error, the loader will issue this diagnostic.
3. CZCDL010 UNNAMED CSECT ASSIGNED PRIVATE STORAGE	Since unnamed CSECTs are assigned unique names within each task, it is impossible for the loader to correlate such names across tasks. The user must, therefore, name his public CSECTs.
4. CZCDL012 PROCEEDING: ILLEGAL ENTRY NAME [x] IN MODULE [a]	This diagnostic arises from several possible sources: 1. An SYSxxxxx symbol appearing in a non-system control section. 2. See Figure 19, column 5, for a complete list of naming restrictions.
5. CZCDL013 PROCEEDING: ENTRY POINT [x], MODULE [a] DUPLICATES CSECT NAME IN MODULE [b]	Self explanatory; the symbol, x, is rejected.
6. CZCDL020 PROCEEDING: ENTRY POINT [x], MODULE [a] DUPLICATES ENTRY POINT IN MODULE [b]	The symbol, x, already exists within the searched DEF chain; the symbol is rejected.
7. CZCDL006 PROCEEDING: PUBLIC CSECT [x], MODULE [a]. UNRESOLVED ADCONS	The loader has encountered a text page within a public CSECT that has adcons. This is not allowed. The adcons on this page will never be relocated if the user decides to proceed.

DIAGNOSTIC	COMMENT
8. CZCDL014 PROCEEDING: CSECT [x] IN MODULE [a] REJECTED, DUPLICATE ENTRY POINT IN MODULE [b]	The loader has rejected the control section named x because x already exists in the DEF chain as an entry point (not a CSECT name).
9. CZCDI018 PROCEEDING: {COMMON / NONCOMMON} CSECT [x], MODULE [a] {COMMON / NONCOMMON} REJECTED BY CSECT, MODULE [b]	If a common section is rejected by the prior occurrence of a noncommon section of the same name (or vice versa), the loader issues this diagnostic.
10. CZCDI017 PROCEEDING: {NONPRIVILEGED / NON-READ-ONLY} CSECT [x], MODULE [a] {REJECTED} BY {PRIVILEGED / READ-ONLY} CSECT, MODULE [b]	This diagnostic results when either a non-read-only CSECT is rejected by a read-only CSECT or a nonprivileged CSECT is rejected by a privileged CSECT. For example, if some user CSECT A is rejected by some read-only CSECT A, any attempt by the user to store into A will result in a storage protection error.
11. CZCDL016 TERMINATED: PRIVILEGED CSECT [x], MODULE [a] REJECTED BY NONPRIVILEGED CSECT, MODULE [b]	If a privileged CSECT name x is rejected by the prior occurrence of a nonprivileged CSECT name x, this diagnostic results. This situation could result in user code being executed in PSW key 0, so the loader initiates ABEND procedures immediately.
12. CZCDL015 PROCEEDING: LENGTH OF REJECTED CSECT [x], MODULE [a] EXCEEDS LOADED CSECT OF MODULE [b]	If the length of a rejected CSECT exceeds the length of the CSECT causing rejection, this diagnostic is issued. Since the user may possibly refer to that segment of the rejected CSECT that lies beyond the upper limit of the already loaded CSECT, there exists the potential for either a storage protection error or an erroneous reference into another CSECT or storage block.
13. CZCDL003 UNDEFINED REF [x] IN MODULE [a]. ADDRESS FFFFF000 ASSIGNED	The symbol x cannot be defined for this task. It does not exist in the libraries searched, was contained in a rejected CSECT, or exists in the wrong hash table. A nonreferable segment and page address is assigned.
14. CZCDL008 PROCEEDING: REF [x] IN MODULE [a] REFERS TO UNDEFINED COMPLEX DEF, MODULE [b]	One situation that might produce this message is: Module A has a complex DEF, B which has a REF to some symbol. There is discovered some REF that refers back to complex DEF R, still undefined.
15. CZCDL007 PROCEEDING: COMPLEX DEF [x] IN MODULE [a] DEFINED AS COMPLEX DEF [y], MODULE [b]	In this case, some complex DEF x has a REF to complex DEF y which is defined in another module at the time the reference is made. This warning is issued even though the REF is properly satisfied. Should the two modules that contain the REF and DEF by link-edited, attempts to load the combined module will result in diagnostic 14 for that same REF.

DIAGNOSTIC	COMMENT
16. CZCDU001 CANCELLED: ARGUMENT SYMBOL FOR DELETE [x] NOT FOUND	The unloader is unable to find the symbol x in the TDY. Symbol x is that symbol which occurred in the operand field of either an assembled DELETE statement or a command language UNLOAD statement. This is issued by UNLINK (CZCDU1) and is accompanied by a return code of 4.
17. CZCDU002 CANCELLED: MODULE DEFINING SYMBOL [x] NOT UNLOADED - OUTSTANDING REFERENCES	The primary deletion candidate; that is, the module that contains the symbol x named in a DELETE macro instruction or UNLOAD statement, was not, in fact, unloaded because of outstanding implicit references remaining to that module after the candidate list was constructed. This is issued by UNLINK (CZCDU1) and is accompanied by a return code of 8.
18. CZCDL004 CSECT [x] IN MODULE [a] REJECTED - ILLEGAL FORM OF CSECT NAME	Refer to diagnostic #4 for explanation of illegal forms.
19. CZCDL001 PROCEEDING: SYMBOL [x] IN LIBRARY [a] NOT OBJECT MODULE OR ENTRY POINT	The loader could not find symbol x defined in a valid object module. This diagnostic message is issued by LIBE SEARCH (CZCDL3) and can be further explained with the EXPLAIN command.
20. CZCDL005 PROCEEDING: MODULE [x] PRODUCED WITH LEVEL [A] ERRORS	Loader found that module was produced with errors: 1 = minor errors 2 = major errors
21. CZCDL009 PROCEEDING: MODULE [x] IS {ILLEGAL / DUPLICATE}, STANDARD ENTRY POINT NOT DEFINED	Module name was either illegal or duplicate, a standard entry point is not defined.
22. CZCDL021 PROCEEDING: CSECT [x], MODULE [a] DUPLICATES A CSECT IN MODULE [b]	The loader has rejected a control section named [x] because x already exists in the DEF chain as a CSECT name.
23. CZCCD201 CANCELLED: MODULE [x] NOT UNLOADED. OUTSTANDING REFERENCES.	The named module could not be unloaded when doing an ERASE, RELEASE, or a DELETE of the library from which the module was loaded. Another module (which was not unloaded) contains a reference to an external symbol in the named module.
24. CZCDL022 PROCEEDING: CONFLICTING ALIGNMENT OR LENGTH WITH DXD [x]	Q-CHAIN (CZCDL7) has encountered DXD instructions with identical names but conflicting lengths or alignments.
25. CZCDL023 PUBLIC CSECT [x] in MODULE [a] ASSIGNED PRIVATE STORAGE	The loader cannot assign public storage to the control section named [x] in module [a] because x's text length is greater than one segment (256 pages). Private virtual storage is assigned.

Where more than one page number is indicated, the major reference is first.

Dynamic Loader Printed in U.S.A. GY28-2031-3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

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