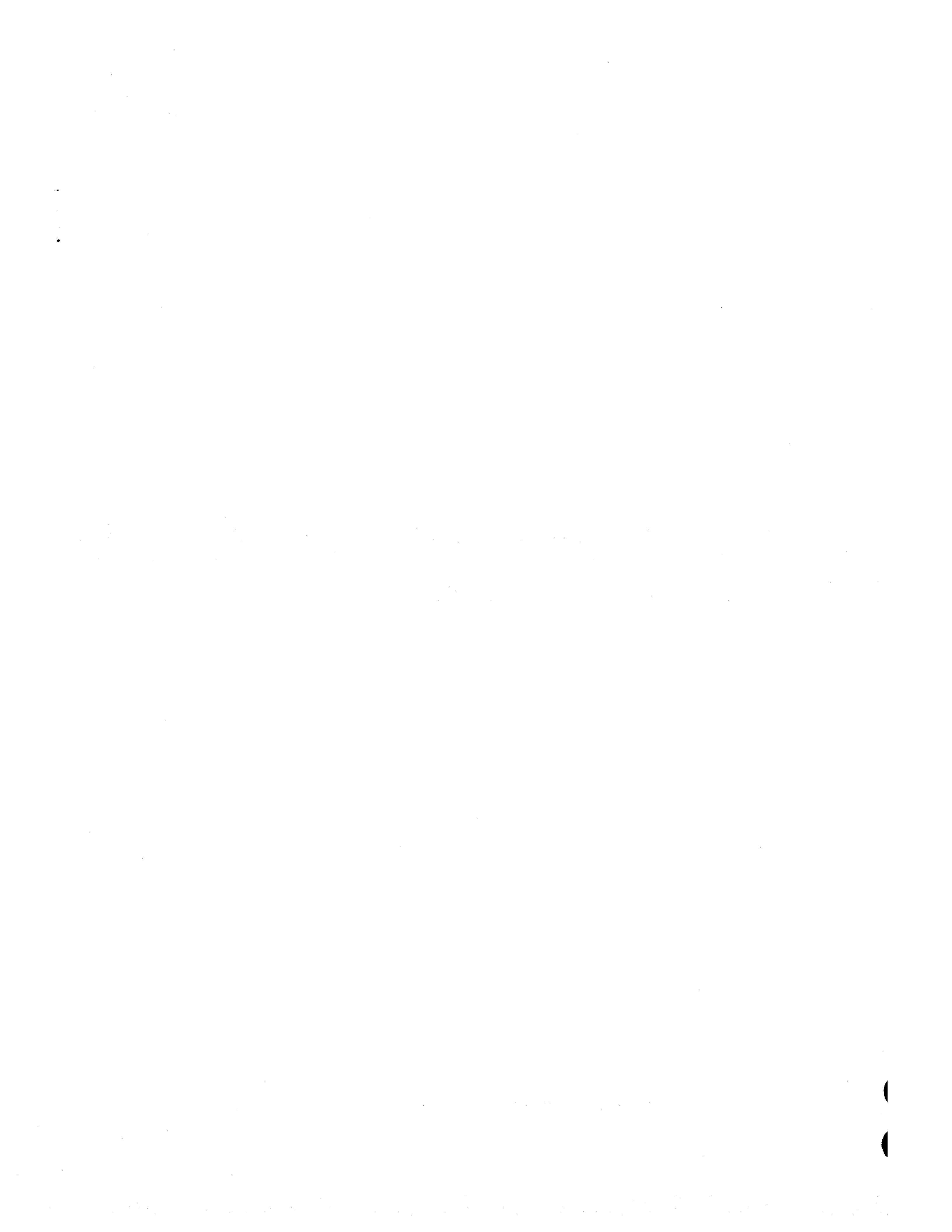
60499200

CONTROL DATA CORPORATION

COMMON MEMORY MANAGER VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 1
NOS 2
NOS/BE 1
SCOPE 2



60499200

CONTROL DATA CORPORATION

COMMON MEMORY MANAGER VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

- NOS 1
- NOS 2
- NOS/BE 1
- SCOPE 2

REVISION RECORD

Revision	Description
A (12/06/76)	Original release. PSR level 439.
B (08/29/77)	Revised to correct technical errors and improve clarity of text. This edition obsoletes the previous edition. PSR level 439.
C (03/31/78)	Revised to reflect Version 1.1 including CMM.FAF, flexible-allocate-fixed, and CMM.LOV, load overlay via FOL.
D (02/15/80)	Revised to add SCOPE 2 support. This revision supersedes all previous editions.
E (10/24/80)	Revised to add FORTRAN calls to CMM. Calls to CMM from other languages are now possible. This revision supersedes all previous editions. PSR level 527.
F (02/26/82)	Revised to correct technical errors and add NOS 2 support. This revision supersedes all previous editions. PSR level 552.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

©COPYRIGHT CONTROL DATA CORPORATION
1976, 1977, 1978, 1980, 1982
All Rights Reserved
Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. BOX 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Title Page	-
ii	F
iii/iv	F
v	F
vi	F
vii	F
1-1 thru 1-6	F
2-1	E
2-2 thru 2-13	F
3-1 thru 3-5	F
A-1	E
A-2	E
B-1 thru B-3	F
C-1	D
Index-1	F
Index-2	F
Comment Sheet	F
Mailer	-
Back Cover	-

PREFACE

As described in this publication, Common Memory Manager Version 1 operates under control of the following operating systems:

NOS 1 for the CONTROL DATA® CYBER 170 Computer Systems; CYBER 70 Models 71, 72, 73, 74; and 6000 Computer Systems

NOS 2 for the CONTROL DATA® CYBER 170 Computer Systems; CYBER 70 Models 71, 72, 73, 74; and 6000 Computer Systems

NOS/BE 1 for the CDC® CYBER 170 Computer Systems; CYBER 70 Models 71, 72, 73, 74; and 6000 Computer Systems

SCOPE 2 for the CDC® Model 176, CYBER 70 Model 76, and 7600 Computer Systems

Common Memory Manager (CMM) provides dynamic memory management as an integral part of the NOS 1, NOS 2, NOS/BE 1, and SCOPE 2 operating systems. CMM is intended as an enhancement to program performance.

CMM can be used with any program that requires memory management and coexists with any independent product which also requires memory management. CMM can also be used in any program that interfaces with product set members which use CMM. The use of CMM is not necessary in any product that has its own memory manager or that does not require memory management.

The user need not be aware that CMM is being used. In most cases, no statements applicable to CMM need be included in the job stream.

NOTE

Continued use of Common Memory Manager by application program jobs should be avoided when the option to do so exists. Program dependence on Common Memory Manager use can complicate migration of a program to a future system.

The product set members that currently use CMM are:

BASIC 3.2 and subsequent versions

COBOL 4 to 5 Conversion Aids

COBOL 5 and subsequent versions

CYBER Database Control System 1.2 and subsequent versions

CYBER Database Control System 2 and subsequent versions

CYBER Loader 1.3 and subsequent versions

CYBER Record Manager Advanced Access Methods 2.0 and subsequent versions

CYBER Record Manager Basic Access Methods 1.5 and subsequent versions

Data Base Utilities 1.2 and subsequent versions

FORM 1.1

FORTRAN Extended 4.7 and 4.8

FORTRAN 5

FORTRAN Common Library 4.7 and 4.8

FORTRAN Common Library 5

PL/I 1.0 and subsequent versions

QUERY UPDATE 3.2 and subsequent versions

Sort/Merge 4.6 and subsequent versions

The reader is assumed to be familiar with the computer system in use, as well as with the operating system, machine assembly language, and loader.

All numbers shown throughout this manual are decimal except where otherwise noted by the use of subscripts. Positive quantities are frequently shown in bit fields longer than possible values, meaning that right-justification is assumed and leading zeros are included. Leading zeros must be present for all input to CMM and are present in all output from CMM.

Detailed information can be found in the listed publications. The publications are listed alphabetically within groupings that indicate relative importance to readers of this manual.

The NOS Manual Abstracts and the NOS/BE Manual Abstracts are instant-sized manuals containing brief descriptions of the contents and intended audience of all NOS and NOS product set manuals, and NOS/BE and NOS/BE product set manuals, respectively. The abstracts manuals can be useful in determining which manuals are of greatest interest to a particular user. The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

Additional information can be found in the listed publications.

The following manuals are of primary interest:

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1	SCOPE 2
COBOL Version 5 Reference Manual	60497100	X	X	X	
COMPASS Version 3 Reference Manual	60492600	X	X	X	X
CYBER Loader Version 1 Reference Manual	60429800	X	X	X	
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700	X	X	X	
FORTRAN Version 5 Reference Manual	60481300	X	X	X	X
SCOPE 2 Loader Reference Manual	60454780				X
SCOPE 2 Record Manager Reference Manual	60454690				X
SYMPL Version 1 Reference Manual	60496400	X	X	X	X
UPDATE Version 1 Reference Manual	60449900	X	X	X	X

The following manuals are of secondary interest:

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1	SCOPE 2
Manual Abstracts for NOS Version 1 Operating System and Product Set	84000420	X			
Manual Abstracts for NOS Version 2 Operating System and Product Set	60485500		X		
Manual Abstracts for NOS/BE Operating System and Product Set	84000470			X	
Software Publications Release History	60481000	X	X	X	X

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

1. GENERAL DESCRIPTION 1-1

Using CMM 1-1
CMM Features 1-2
- Static Area 1-2
- Dynamic Area 1-3
  - Dynamic Area Base Address Pointer 1-4
  - Dynamic Area Header and Trailer 1-4
- Blocks and Block Groups 1-4
  - Fixed Block Header 1-5
  - Block Group Pointers 1-5
- Free Space 1-6
- CMM Internal Storage 1-6

Control of CMM 1-6

2. CMM OPERATION AND USE 2-1

Fixed-Position Block Calls 2-1
- Allocate or Create Fixed-Position Blocks 2-1
- Flexible Allocate 2-2
- Free or Destroy Fixed-Position Blocks 2-3
- Shrink Fixed-Position Blocks 2-3
  - Shrink at fwa 2-3
  - Shrink at lwa 2-3
- Grow Fixed-Position Blocks 2-4
- Change Fixed-Position Block Specifications 2-4
- Save Identified Value 2-5
- Get Block Address 2-5

Overlays, Segments, and the Static/Dynamic Area Boundary 2-5
- Load Overlay 2-6
- Load Overlay via FOL 2-6
- Respecify the Dynamic Area 2-7
- Respecify the Highest High Address 2-7
- Deactivate CMM 2-8

Block Group Calls 2-8
- Activate a Block Group 2-8
- Free a Block Group 2-8

Information Calls 2-9
- Get Block Information 2-9
- Get Maximum Available Fixed Block Size 2-9
- Get Statistics for Overflow Recovery 2-10
- Get Statistics for Job Summary 2-10

Efficiency-Increasing Calls 2-12
- Optimization Functions 2-12
  - CMM.OP1 Function 2-12
  - CMM.OP2 Function 2-12
  - CMM.OP4 Function 2-12
- Set-Own-Code Error Processing 2-12

Low Memory Communication Words 2-13
CMM Use 2-13
- CMM Selection 2-13
- Legal First Calls 2-13

3. CMM INTERFACE TO COMPILERS 3-1

FORTRAN Version 5 Interface to CMM 3-1
SYMPL Interface to CMM 3-2
COBOL Version 5 Interface to CMM 3-2

APPENDIXES

A Standard Character Sets A-1
B Diagnostics B-1
C Glossary C-1

INDEX

FIGURES

1-1 Illustrated CMM Terminology 1-2
1-2 Dynamic Area - Typical Layout 1-3
1-3 CMM Active Format 1-4
1-4 Dynamic Area Header Format 1-4
1-5 Dynamic Area Trailer Format 1-4
1-6 Fixed Block Header 1-5
1-7 Block Group Pointer 1-5
1-8 Free Space Header 1-6
2-1 Fixed Block Adjustments 2-1
2-2 Dynamic Area Statistics 2-11
2-3 Register X1 Contents 2-12
3-1 FORTRAN/CMM Example Referencing the Block as an Array 3-1
3-2 FORTRAN/CMM Example Referencing With an Offset 3-1
3-3 SYMPL/CMM Interface Example 3-2
3-4 COBOL/CMM Interface Example 3-2
3-5 Sample COBOL/CMM Program 3-4

TABLES

2-1 CMM.ALF Entry Conditions 2-2
2-2 CMM.FAF Entry Conditions 2-3
2-3 CMM.SIV Entry Conditions 2-5
2-4 CMM.FWA Exit Conditions 2-5
2-5 CMM.LOV Entry Conditions 2-7
2-6 CMM.LOV Exit Conditions 2-7
2-7 CMM.GBI Exit Conditions 2-9

Common Memory Manager (CMM) is composed of a set of intercommunicating and relocatable central processor routines contained in the principal system library. These routines are loaded into the user field length during the time external references are being satisfied. CMM provides centralized dynamic memory management for application programs as well as for the CYBER 170 and CYBER 70 product set. Users of data base management programs, such as CYBER Record Manager, can utilize the CMM facilities provided by those programs.

Centralized dynamic memory management opens up many new possibilities for dealing with memory size problems. With memory management, the dynamic loading of infrequently executed code and/or the moving of data areas to slow memory (either extended core storage (ECS) or large core memory (LCM)) becomes more efficient. Also, when CMM is combined with the operating system field length assignment algorithm, the product set becomes significantly easier to use at the job control level because field length management is automatic, and program execution time is decreased.

Simple addressing errors generated by running programs can cause program failure. Such addressing errors, although repeatable, can be hard to correct because the failure is detected much later than the actual occurrence of the error. Also, the detected failure can exhibit symptoms unrelated to the nature of the actual error. Therefore, CMM has been implemented in two versions: a fast version, and an error-checking version. The error-checking version is intended for program checkout only; all user-specified parameters are checked. The fast version does no error checking; it is intended for subsequent use when no user errors remain. The desired CMM version can be selected through the use of system assembly options.

USING CMM

The CMM routines are a set of CP COMPASS subroutines which are loaded into the user's field length. By calling these routines with various parameters, operations such as allocating space, releasing space, changing the size of blocks previously allocated, and obtaining various memory management statistics can be performed.

CMM depends on several pointers in memory to perform its operations. The major pointer, memory cell 65_8, is off-limits to all products except CMM and the loader. After loading, the loader sets ra+65 to lwa+1 of the load. At the first call to one of the CMM routines, CMM complements ra+65 and uses this address as the first word of the CMM dynamically managed area. This header word has been given the name daba (dynamic area base address). By complementing ra+65, CMM can determine if a call is the first CMM call or if CMM has been called before, and the various products and users can check this memory cell to see if CMM is managing memory space.

For a non-overlayed program, daba is a constant value throughout processing. With overlayed programs, however, daba changes value as each overlay is loaded. All overlay load requests must be made via CMM so that any dynamic blocks that might reside in space that an overlay would occupy can be protected. To avoid conflict, the loader routines that load overlays ensure that non-CMM loader requests are aborted when CMM is active.

The fact that various overlays are different lengths, and that daba changes, leads to two types of possible CMM blocks. A safe block is one type. It is allocated above highest high address (hha) where it is beyond reach of the largest overlay that might be loaded. An unsafe block is the other type. It must be released before an increase in daba occurs. Unsafe blocks are especially useful for storing transient data or routines that can be freed before another overlay is loaded. The product set uses only safe blocks to prevent any restrictions on overlay usage by the general user. CYBER Record Manager (CRM), under NOS and NOS/BE, is an exception. CRM has implemented a FIT field known as BBH that can be set to YES (the default is NO). If this field is set and a dynamic block is allocated for buffer space, CRM will request CMM to allocate the block below hha if enough space is available. The user must make sure the file is closed appropriately so that the buffer is released before loading a larger overlay into memory. Use of the BBH field is further described in an appendix of the CRM Basic Access Methods reference manual. The BBH field is not available under SCOPE 2 Record Manager.

These unsafe blocks are always assigned as part of a group of blocks so that CMM can monitor them as a unit when overlays are being loaded. CMM contains a call that allows the user to free all the blocks in the group without having to free them individually. For instance, by specifying several unsafe blocks as local to an overlay, all of them can be unloaded without having to release each block individually.

Independent of the placement of the CMM blocks is the type of CMM block that can be allocated. The usual (and simplest) type of block that is requested is a fixed-position, fixed-size block. The user requests a block of a specific size, and CMM returns the first word address for that block. Other attributes associated with the size of the block can be requested when the block is first obtained. These attributes include the ability to reduce the starting address of the block, or reduce or extend the ending address of the block. The latter block is also known as a grow-end block, and should be avoided because no blocks are allowed to reside in memory at a higher address than the grow-end block. (If CMM were to allow another block above the grow-end block, then the possibility of extending the grow-end block into the new block would exist.)

After a CMM block is allocated and passed on to the user, it can be used like any other piece of memory. The only cautions are that if the bounds of the memory block are exceeded, it will destroy CMM's internal pointer chain. Also, any memory preset in the block is the responsibility of the user since CMM passes the block to the user with memory contents undefined.

CMM subroutine calls exist to manipulate the blocks in limited fashion after they are allocated. There are calls to shrink the first word address (fwa) and the last word address (lwa) of blocks that were acquired with this characteristic. Also, a call exists to grow the end of a block that was acquired with grow-end characteristics.

(Grow-end blocks are not recommended and should be avoided.) A call is also available to change the characteristic of a block from shrink-lwa or shrink-fwa or grow-end to fixed position, fixed size.

After using a CMM block, a call is available to release the block. Only the first word address of the block is required; CMM does the rest.

CMM provides several calls to change the pointers it uses during processing. These calls allow the user to change daba or hha in the program. For instance, if an application starts out with a very large primary overlay, and after that overlay is processed it is not called again for the duration of the job step, a call to the CMM routine could change hha from the end of the largest overlay to the end of the next largest overlay. CMM could then allocate global blocks in the portion of memory formerly used by the largest overlay.

Other features of CMM include calls that can be made to obtain information about any particular block, such as its attributes or size. Calls are also available to find the size of the largest possible fixed-position block that CMM can assign. There are two variants to this call: one determines the biggest block that does not involve a job field length increase, and the other determines the biggest block that is possible with a field length increase. Calls also are available for statistics to aid in determining what kind of overflow action to take if memory is not available for a large block. For example, a call for summary statistics returns the number of field length increases and decreases and the number of blocks obtained and released.

CMM overhead is quite low. When the 13 modules in CMM are all in use, they do not require more than 1200_8 words. Only the routines being used are loaded; these usually require less than 400_8 words.

CMM FEATURES

Terms used in describing the features of CMM are defined in the paragraphs and figures that follow. Low memory, move down, and so forth, always refer to the reference address (ra) or lower end of the field length (fl). High memory, move up, and so forth, refer to the upper end (ra+fl) of the field length. In all descriptions of table entries, word 0 refers to the first word of the table or entry, word 1 refers to the second, and so forth.

Initial load or initial loading refers to that loading which is determined by the nature of the job step and which starts at the beginning of the job step. Three forms of initial loading are currently supported:

Basic (relocatable)

Overlay

Segment

User-call loading is not considered to be initial loading because the address space occupied by the loaded material is not mapped out prior to the beginning of job step execution. The action of loading those overlays formed in a different overlay generation sequence than the initially loaded (0,0) overlay is considered to be a special case, called extended overlay loading, as explained in section 2.

STATIC AREA

The static area is shown in figure 1-1. This area originates at the reference address and extends high enough in memory to contain all of the addresses mapped out by the loader when the loader constructed the portion of the

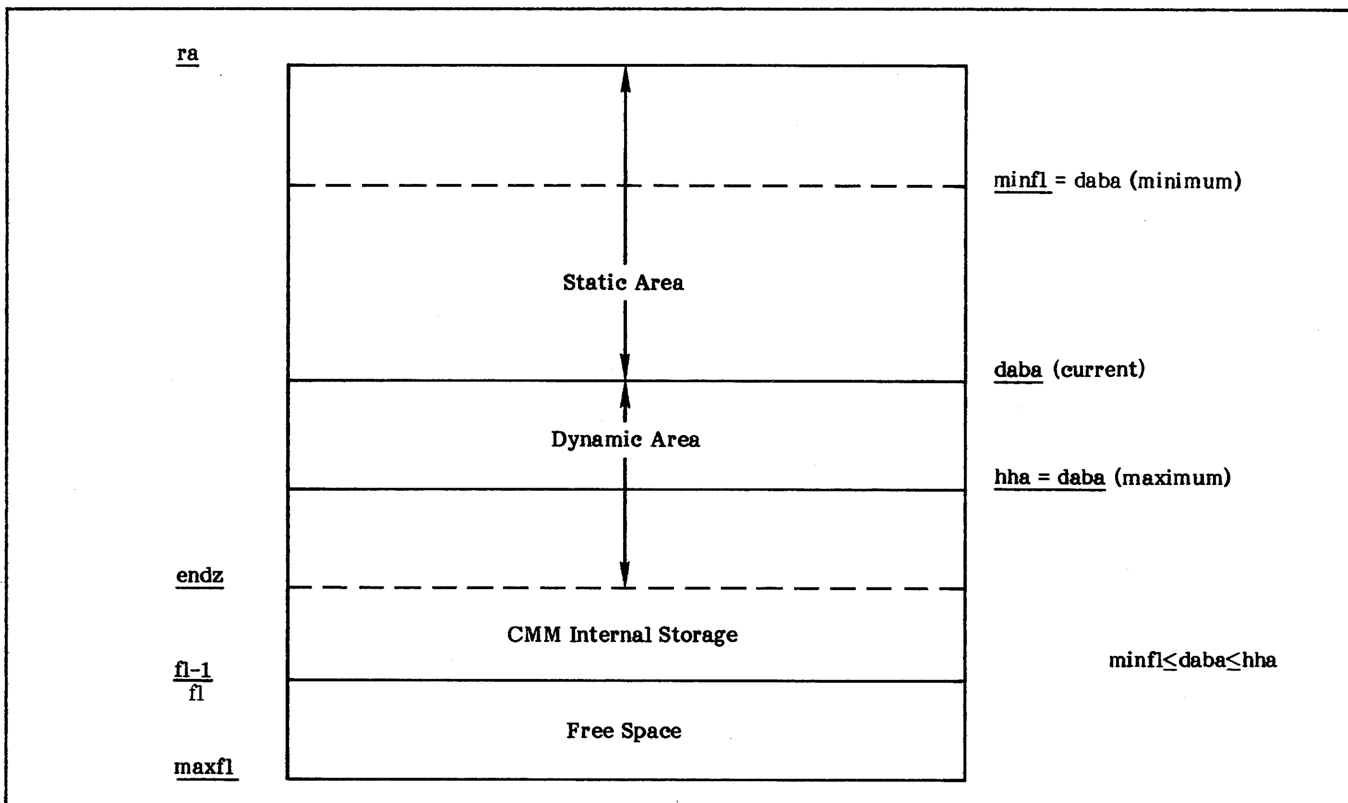


Figure 1-1. Illustrated CMM Terminology

initial load now in memory, whether or not that area was preset. Basic initial loading establishes a static area that remains fixed in size throughout the job step, as contrasted with overlay and segment loading, which establish a static area that can fluctuate in size as the job step proceeds.

The term minfl denotes the current length of the static area; the actual location to which minfl points is the first word past the static area. The highest high address (hha) denotes the largest value of minfl for any portion of the initial load; that is, hha equals minfl for basic loads, but hha is increased the length of the longest overlay or the longest path through the tree for segments. The term daba refers to the dynamic area base address and is used for a number that communicates the current value of minfl to CMM. Under normal circumstances, daba equals minfl; however, the only constraint required by CMM is that minfl be less than or equal to daba, which in turn is less than or equal to hha. This relationship is illustrated in figure 1-1.

DYNAMIC AREA

The dynamic area is that portion of the field length starting at address daba and continuing to the current fl.

The term maxfl denotes the maximum field length value permitted to the job step as a function of job, operating system, and installation parameters. A typical layout of the dynamic area is shown in figure 1-2.

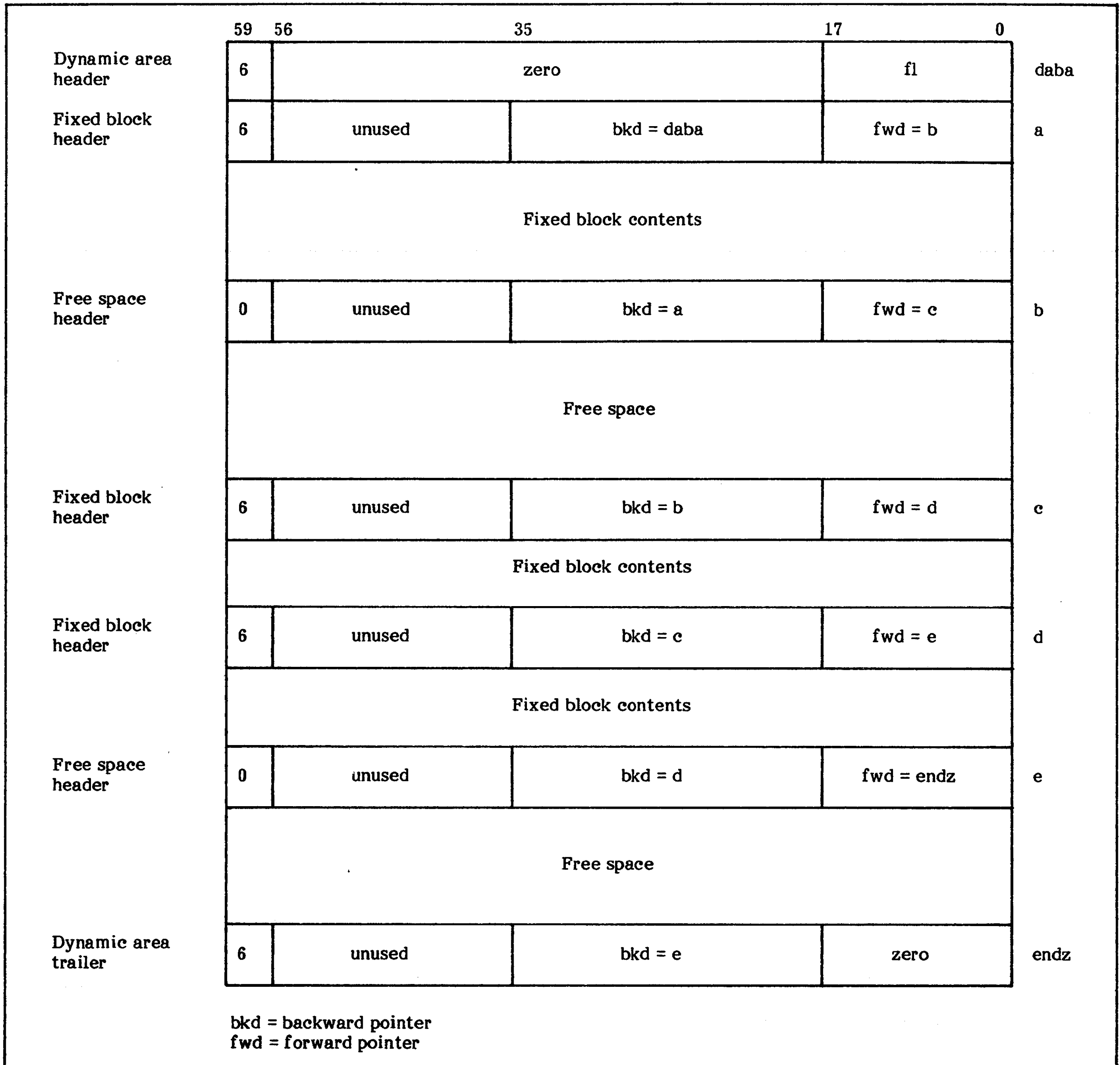


Figure 1-2. Dynamic Area – Typical Layout

All of CMM's internal storage, except for items local to a single module, extends from fl-1 downward in memory to endz. The area from fl upward to maxfl is considered to be available free space as far as the computing utilization level is concerned, even though this area is not part of the field length.

Dynamic Area Base Address Pointer

The base address of the dynamic area (daba) changes with each loading of a job step. The value of daba cannot be less than minfl nor more than hha. The current value of daba, in complement form, is indicated in word ra+65_8, which, once CMM is activated by a call to one of its functions, has the format shown in figure 1-3.

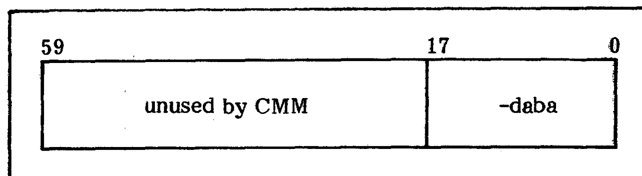


Figure 1-3. CMM Active Format

Dynamic Area Header and Trailer

The header and trailer words, which denote the dynamic area, both have a type code 6 in bits 59 through 57. The fixed-block header also has a type code 6. (See figure 1-2.)

The format of the dynamic area header is shown in figure 1-4.

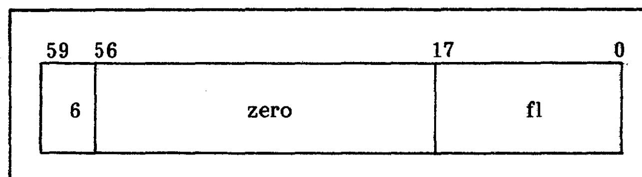


Figure 1-4. Dynamic Area Header Format

In figure 1-4, fl is the current field length, which contains the static area, dynamic area, and CMM internal storage area.

This header word designates the dynamic area; its location is known as the dynamic area base address, or daba. A pointer word in ra+65_8 contains the complement of daba. (See Dynamic Area Base Address Pointer.)

The format of the dynamic area trailer is shown in figure 1-5.

In figure 1-5, bkd is the backward pointer to the last header in the dynamic area.

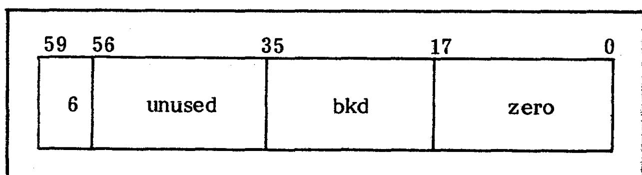


Figure 1-5. Dynamic Area Trailer Format

BLOCKS AND BLOCK GROUPS

A block is a portion of memory within the field length, identified logically and required to be contiguous for addressing reasons. CMM is called to obtain blocks whose position remains fixed. These blocks are of arbitrary size subject to the maximum field length limitation.

All blocks occupy a position that is determined by CMM. This position is communicated to the requesting program via register X1. There is no provision for any user control over actual block position. All blocks, once obtained, remain active until explicitly freed by a CMM call. The entire contents of a block, involving an integral number of words and possibly involving block size changes, are available to the user during the active lifetime of the block. These contents remain completely unmodified by CMM. No word of memory in the dynamic area, located outside of the active block, can be modified by any program other than CMM when CMM is active.

It is often useful to cluster independently obtained blocks into logical groups for purposes of being freed. The most common uses for the freeing capability implemented in CMM are:

> When unloading an overlay or segment, it is usually necessary to free the dynamic space obtained by the programs being unloaded.
>
> When recovering after an error in a subsystem such as CYBER Record Manager, it is useful to be able to discard (free) all dynamic space obtained by that subsystem.

A particular call to CMM activates a logical block group; a unique integer identifier for the block group is returned to the user. This identifier can be used when obtaining a block, thus specifying to CMM that the obtained block is a member of the identified group. Another call to CMM deactivates a logical block group and frees all blocks belonging to it. A block that belongs to a group can also be freed on its own. All blocks having the same identifier are chained together by forward and backward pointers as illustrated later in this section. A group remains active even if all blocks belonging to it are individually freed.

The group concept is also used in CMM to solve the problem which arises from usage of that portion of the dynamic area below hha. CMM is generally unaware of the circumstances that cause the static area to change size. Without information on the lifetime of blocks, CMM cannot place any fixed-position blocks below hha. Circumstances exist where this area constitutes the majority of the available dynamic area, and where the user recognizes the fact that certain obtained blocks have a lifetime shorter than the next decrease in the area.

In CMM, the solution to this problem is to establish two types of groups, called type 0 and type 1. Safe blocks can only remain safe as members of type 0 groups. (Type 0 groups usually contain only safe blocks.) Any block allocated as a member of a type 1 group must have a lifetime shorter than the next increase in the size of the static area. This is enforced by requiring that no type 1 groups be active when the static area is increased in size. Type 1 groups contain only unsafe blocks. Blocks allocated as members of a type 1 group are preferentially (but not necessarily) allocated below hha. Other fixed-position blocks are never allocated below hha.

Fixed Block Header

Fixed blocks are located in the dynamic area. Although the size of fixed blocks can change, their location in the dynamic area remains fixed. Each block has a 1-word or 2-word header of the format shown in figure 1-6.

Block Group Pointers

All fixed blocks that are members of a block group having a common identifier are chained together by forward and backward pointers. The chain begins in the group-id entry, which has the format shown in figure 1-7. Blocks in the group are not chained together in any special order. The fixed-block pointers are in word 1 of the header, which is present only if the block is a member of a block group.

Word	59	56 53	47	44	43	35	17 0
0	6	zero	sc	n	unused	bkd	fwd
1	25	gid				gbkd	gfwd

sc Size code given in three bits:

- 1xx fwa end can shrink
- x1x lwa end can shrink
- xx1 lwa end can grow

n Flag bit set to 1 if word 1 is present; the block is then part of a block group.

bkd Backward pointer to the previous region; bkd points to word 0 of the header.

fwd Forward pointer to the next region; fwd points to word 0 of the header.

gid Value of the group-id.

gbkd Pointer to a previous block having the same group-id; gbkd points to word 1 of the header.

gfwd Pointer to the next block having the same group-id; gfwd points to word 1 of the header.

Figure 1-6. Fixed Block Header

59	58 53	35	17	0
gt	unused	gid	zero	gfwd

gt Group type:

- 0 Unrestricted lifetime
- 1 Lifetime limited to the next increase in daba

gid Group-id.

gfwd Pointer to the header of the first block in the chain having this group-id.

Figure 1-7. Block Group Pointer

FREE SPACE

Free space within the dynamic area is identified by a free space header. This header has the same format as word 0 of the fixed block header; however, it contains zero in bit positions 57 through 59. The free space header is shown in figure 1-8.

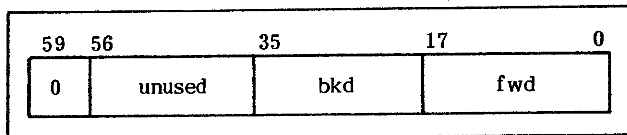


Figure 1-8. Free Space Header

CMM INTERNAL STORAGE

The area in which CMM maintains a variety of pointers, flags, subroutines, and stacks is located between the dynamic area and the upper boundary of the field length as shown in figure 1-1. The storage area extends downward in memory from fl-1 to endz, a location which follows the end of the dynamic area. The upper end of the area contains pointer words to each block in the lower part of the storage area. The pointer words contain the actual address of each block and its length in number of words.

Five types of blocks can be maintained in the internal storage area. The types are:

1 Bootstrap code

2 Reserved for future use

3 Reserved for future use

4 Identified values

5 Group-id entries

CONTROL OF CMM

It is important that the division of control between the user and CMM be understood. CMM is just a service routine that responds to various user requests; however, certain restrictions are placed on the user regarding the use of the dynamic area. The interface between CMM and the user is carefully defined so as to draw a definite and discrete line between what the user controls and what CMM controls.

CMM can be called to obtain an arbitrary number of blocks having various properties and sizes, subject to maxfl limitations. Each block occupies a position determined by CMM and communicated to the user via systematic convention. There is no provision for any user control over actual block position. All blocks, once obtained, remain active until explicitly freed by a CMM call.

During the active lifetime of a block (possibly involving block size variations), its entire contents, including an integral number of words, are available to the user. These contents remain completely unmodified by CMM.

When CMM is active, no word of memory that is in the dynamic area, but outside of an active block, can be modified by any program except CMM. For example, all non-CMM loader requests are aborted while CMM is active.

CMM can be called to change the effective size of the static area by changing the dynamic area base address (daba). This is done, for example, when overlays or segments are loaded. However, since daba must always remain less than or equal to the highest high address (hha), all addresses from hha up are always part of the dynamic area.

CMM manages the total size of the dynamic area in a completely user-transparent fashion by managing the current space which falls between the limits of hha and maxfl. Requests to the operating system for field length change must not be made by any program other than CMM when CMM is active.

The user, therefore, has explicit control over the number, size, lifetime, and contents of all active dynamic blocks, and also has bounded control over the size of the static area. The user has no control whatsoever over the position of any dynamic block, the contents of any portion of the dynamic area outside of the active blocks, or any direct control over the size of the dynamic area.

Common Memory Manager (CMM) is composed of several relocatable routines. Certain of these routines are resident whenever CMM is in use. These resident routines are referenced by all of the CMM functions and perform such actions as changing field length, managing the internal tables of CMM, computing the utilization level, and initializing the dynamic area.

CMM routines can be called from COMPASS or from one of several other languages such as FORTRAN, COBOL, or SYMPL. The logical form used by COMPASS is described first in the following descriptions of the CMM functions. The FORTRAN form of the call is second. The COMPASS entry point and register conventions are also included. If the COMPASS call is not used, the calling language must follow the FORTRAN calling sequence conventions. All parameters must be 60-bit integer items aligned on word boundaries. Additional information on calling CMM from FORTRAN, SYMPL, and COBOL appears in section 3. Use of languages other than COMPASS or FORTRAN requires that the user be aware of how the language assigns arrays because, in many cases, the interface between the calling language and CMM is laid out in the form of an array. (Refer to Set-Own-Code Error Processing later in this section for an example.)

Under NOS and NOS/BE, all of the language callable routines reside in the library SYMLIB. Therefore, the CYBER loader directive LDSET(LIB=SYMLIB) should be included in the load sequence of all programs referencing them. Under SCOPE 2, these routines reside in the library SYMIO; for SCOPE 2, use LDSET(LIB=SYMIO).

FIXED-POSITION BLOCK CALLS

Calls to CMM are used to allocate fixed-position blocks, free them when no longer needed, change specifications for the block, and alter the size of the block by deleting or adding a specific number of words. Fixed blocks can be made to shrink at either the first word address (fwa) or last word address (lwa) end of the block, or to grow at the lwa end of the block, as illustrated in figure 2-1.

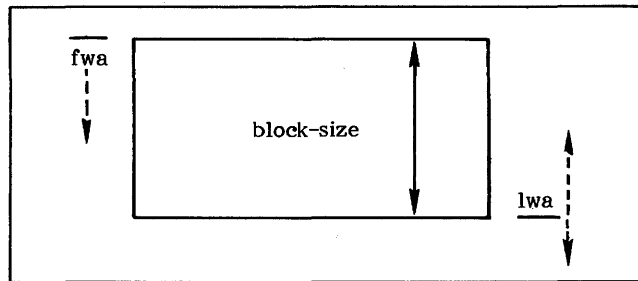


Figure 2-1. Fixed Block Adjustments

ALLOCATE OR CREATE FIXED-POSITION BLOCKS

This call allocates or creates a fixed-position block of a size specified in the call. The first word address (fwa) of the block is returned in register X1. The initial contents of the block are undefined.

The logical form of the call is:

```
allocate-fixed(block-size,size-code,group-id)
returning block-fwa
```

The FORTRAN form of the call is:

```
CALL CMMALF(block-size,size-code,group-id,
block-fwa)
```

block-size — Number of words required for the block

size-code — A 3-bit code indicating the permitted block size variations: 000 indicates fixed; a 1 in any position indicates varying as explained below. Each bit indicates whether a particular end of the block can move; the leftmost bit is used for the first word address end of the block; the second and third bits are used for the last word address end of the block.

- 1xx — fwa end can shrink
- x1x — lwa end can shrink
- xx1 — lwa end can grow

group-id — Zero indicates the block does not belong to a group; a positive integer identifies the active block group to which the block is to be allocated. The integer must be one of the group-id values returned previously by the activate-group call (described later in this section).

block-fwa — First word address of the allocated space for this block. CMM positions each block so that access to eight words after the lwa of the block is made possible without causing an address range error exit.

The fixed block contains exactly block-size words starting at address block-fwa, returned in register X1. The only words in the dynamic area that the user can modify are the words belonging to the active block; the eight addressable words at the end of the block are not to be considered as part of the block and cannot be modified by the user. The contents of the eight words are considered to be undefined.

If a fixed block has a binary size-code of xx1 (indicating the lwa end of the block is permitted to grow), then no other fixed block can be allocated while this block remains active and retains the size-code xx1.

The presence of a fixed block with an extendable lwa end poses a severe restriction on running programs; use of such a block must be a matter of deliberate planning. A user call to the loader, for example, causes such a block to be used; therefore, no other subsystem that coexists with the user-called loader can use such a block.

The COMPASS calling conventions are:

Entry point name:

CMM.ALF (entry is effected by return jump)

Entry conditions are shown in table 2-1.

Exit conditions:

Register X1 contains block-fwa
Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

TABLE 2-1. CMM.ALF ENTRY CONDITIONS

Register	Bits	Contents
X2	59-0	block size
X3	59-29	zero
	28-12	group-id
	11-9	zero
	8-6	size-code
	5-0	zero

FLEXIBLE ALLOCATE

This call is similar to the CMM.ALF function, except that the size of the block to be allocated is specified within a range. Specification of CMM.FAF allows CMM to make more efficient use of the free space.

The logical form of the call is:

flexible-allocate-fixed(min-block-size,max-block-size, size-code, group-id)returning block-fwa and block-size

The FORTRAN form of the call is:

CALL CMMFAF(min-block-size,max-block-size, size-code,group-id,block-fwa,block-size)

min-block-size — Minimum number of words required for the block.

max-block-size — Maximum number of words required for the block.

size-code — A 3-bit code indicating the permitted block size variations: 000 indicates fixed; a 1 in any position indicates varying. Each bit indicates whether a particular end of the block can move; the leftmost bit is used for the first word address end of the block; the second and third bits are used for the last word address end of the block.

1xx	fwa end can shrink
x1x	lwa end can shrink
xx1	lwa end can grow

group-id — Zero indicates the block does not belong to a group; a positive integer identifies the active block group to which the block is to be allocated. The integer must be one of the group-id values returned previously by the activate-group call (described later in this section).

block-fwa — First word address of the allocated space for this block. CMM positions each block so that access to eight words after the lwa of the block is made possible without causing an address range error exit.

block-size — Number of words actually allocated for the block (min-block-size $\leq$ block-size $\leq$ max-block-size).

CMM returns the fwa of the first block it finds that is at least as large as min-block-size. At this fwa, CMM allocates a block with a size as large as possible without exceeding max-block-size. No attempt is made to find a larger block, even if the largest possible block at this fwa is smaller than max-block size.

The first block contains exactly block-size words, returned in register X2, starting at address block-fwa, returned in register X1. The only words in the dynamic area that the user can modify are the words belonging to the active block; the eight addressable words at the end of the block are not to be considered as part of the block and cannot be modified by the user. The contents of the eight words are considered to be undefined.

If a fixed block has a binary size-code of xx1 (indicating the lwa end of the block is permitted to grow), then no other fixed block can be allocated while this block remains active and retains the size-code xx1.

The presence of a fixed block with an extendable lwa end poses a severe restriction on running programs; use of such a block must be a matter of deliberate planning. A user call to the loader, for example, causes such a block to be used; therefore, no other subsystem that coexists with the user-called loader can use such a block.

The COMPASS calling conventions are:

Entry point name:

CMM.FAF (entry is effected by return jump)

Entry conditions are shown in table 2-2.

Exit conditions:

Register X1 contains block-fwa
Register X2 contains block-size
Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

TABLE 2-2. CMM.FAF ENTRY CONDITIONS

Register	Bits	Contents
X2	59-48	zero
	47-30	max-block-size (if zero, min-block is used)
	29-18	zero
	17-0	min-block-size
X3	59-29	zero
	28-12	group-id
	11-6	size code
	5-0	zero

FREE OR DESTROY FIXED-POSITION BLOCKS

This call destroys the fixed-position block whose current fwa is block-fwa. When the block is destroyed, the contents of the block are no longer accessible to the user.

The logical form of the call is:

free-fixed(block-fwa)

The FORTRAN form of the call is:

CALL CMMFRF(block-fwa)

block-fwa — The current first word address of the block to be affected; fwa is obtained from X1 of the allocate/create call return.

The COMPASS calling conventions are:

Entry point name:

CMM.FRF (entry is effected by return jump)

Entry conditions:

Register X1 contains block-fwa

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

SHRINK FIXED-POSITION BLOCKS

Fixed-position blocks can be reduced in size at either the fwa or lwa end.

Shrink at fwa

A specific number of words are deleted from the block at the fwa end, and the contents of the deleted words are lost. If zero is given for the number to be deleted, no change to the block is made.

The logical form of the call is:

shrink-at-fwa-fixed(block-fwa,num)

The FORTRAN form of the call is:

CALL CMMSFF(block-fwa,num)

block-fwa — The current first word address of the block to be affected; must be the fwa of an active fixed-position block.

num — Number of words to be deleted; must not be negative and must not be greater than the current block size.

The current binary size-code of this block must be 1xx, permitting the fwa end of the block to shrink. This call increases the fwa of the block by num.

The COMPASS calling conventions are:

Entry point name:

CMM.SFF (entry is effected by return jump)

Entry conditions:

Register X1 contains block-fwa
Register X2 contains num

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

Shrink at lwa

A specific number of words are deleted from the block at the lwa end, and the contents of the deleted words are lost. If the number of words to be deleted is zero, no change to the block is made.

The logical form of the call is:

shrink-at-lwa-fixed(block-fwa,num)

The FORTRAN form of the call is:

CALL CMMSLF(block-fwa,num)

block-fwa	Current fwa of the block to be affected; must be the fwa of an active fixed-position block.
num	Number of words to be deleted; must not be negative and must not be greater than the current block size.

The current binary size-code of this block must be x1x, permitting the lwa end of the block to shrink.

The COMPASS calling conventions are:

Entry point name:

CMM.SLF (entry is effected by return jump)

Entry conditions:

Register X1 contains block-fwa
Register X2 contains num

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

GROW FIXED-POSITION BLOCKS

Fixed-position blocks can be extended at the last word address end. A specified number of words are added to the block at the lwa end; the contents of the added words initially are undefined.

The logical form of the call is:

grow-at-lwa-fixed(block-fwa,num)

The FORTRAN form of the call is:

CALL CMMGLF(block-fwa,num)

block-fwa	Current fwa of the block to be affected; must be the fwa of an active fixed-position block.
num	Number of words to be added at the lwa end of block; if the number is zero, no change to the block is made. The value of num must be positive.

The current binary size-code of the fixed block must be xx1.

The COMPASS calling conventions are:

Entry point name:

CMM.GLF (entry is effected by return jump)

Entry conditions:

Register X1 contains block-fwa
Register X2 contains num

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

CHANGE FIXED-POSITION BLOCK SPECIFICATIONS

This call permits the current size-code of a fixed-position block to be changed. The allowable fixed block size codes are discussed under the Allocate or Create Fixed-Position Block call earlier in this section.

The logical form of the call is:

change-specs-fixed(block-fwa,new-size-code)

The FORTRAN form of the call is:

CALL CMMCSF(block-fwa,new-size-code)

block-fwa	The current first word address of the block to be affected; must be the fwa of an active fixed-position block.
new-size-code	The new binary size-code to be applied. Unless the code is already xx1, the code must be xx0 or -1; end growth at lwa cannot be added to a block by this call. If the new-size-code is not specified as -1, the current size-code of the block is replaced with new-size-code.

The COMPASS calling conventions are:

Entry point name:

CMM.CSF (entry is effected by return jump)

Entry conditions:

Register X1 contains block-fwa
Register X2 contains new-size-code

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

SAVE IDENTIFIED VALUE

A call to this subroutine causes an arbitrary 30-bit value to be associated with a unique-id specified by CMM, or a user, so as to be accessible via the get-block-address function.

The logical form of the call is:

save-identified-value(value,id)optionally returning unique-id

The FORTRAN form of the call is:

CALL CMMSIV(value,id,unique-id)

value	Any 30-bit value.
id	Zero indicates that CMM is to supply a unique-id. A positive even integer less than 2^{16} is a user-defined unique identifying code. The even values from 2^{16} to 2^{17}-2 inclusive are reserved for CDC.
unique-id	A CMM-defined unique identifying code for the block that is a positive odd integer less than 2^{17}.

The COMPASS calling conventions are:

Entry point name:

CMM.SIV (entry is effected by return jump)

Entry conditions are shown in table 2-3.

Exit conditions:

Register X1 contains odd unique-id if id parameter is given as zero; otherwise, the content of register X1 is undefined

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

TABLE 2-3. CMM.SIV ENTRY CONDITIONS

Register	Bits	Contents
X2	59-30	Zero
	29-0	Value to be saved
X3	59-18	Zero
	17-0	Positive, even, nonzero, unique identifying code; or zero if CMM is to supply unique-id

GET BLOCK ADDRESS

This subroutine makes available to the caller the first word address and current size of a block whose unique identifier is supplied in the call.

The logical form of the call is:

get-block-fwa (unique-id)returning block-fwa and block-size

The FORTRAN form of the call is:

CALL CMMFWA(unique-id,block-fwa,block-size)

unique-id	A positive integer less than 2^{17} saved as unique-id by CMM.SIV.
block-fwa	The 30-bit value saved by CMM.SIV.
block-size	The current length, in number of words, of the identified block.

Provided that the block-fwa has been passed as the 30-bit value in a CMM.SIV call, it is possible to recover the fwa of a fixed block across overlay loads through use of this call.

The COMPASS calling conventions are:

Entry point name:

CMM.FWA (entry is effected by return jump)

Entry conditions:

Register X2 contains unique-id

Exit conditions are shown in table 2-4.

Registers preserved:

A0, X0, B2, B3, X5

TABLE 2-4. CMM.FWA EXIT CONDITIONS

Register	Bits	Contents
X1	59-30	block size
	29-0	block-fwa
B1	17-0	1

OVERLAYS, SEGMENTS, AND THE STATIC/DYNAMIC AREA BOUNDARY

In general, CMM controls the size of the dynamic area. CMM determines the initial extent of the dynamic area without the need for an initializing call. Initially, the loader sets daba equal to minfl, obtains maxfl from the operating system, and, during program execution, manages the job field length in a manner compatible with good system throughput. CMM must, however, be informed of changes in daba and/or hha.

The loading of overlays and segments is the usual cause of changes in daba. In programs using CMM, overlays must be loaded with either 1) the LOADREQ macro specifying the CMM-present parameter, or 2) a call to CMM.LOV if the overlay being loaded is a higher level overlay whose (0,0) overlay contains an FOL (Fast Overlay Loading) directory. CMM.LOV and Fast Overlay Loading are not available under the SCOPE 2 operating system. The LOADREQ macro generates a call to the load-overlay entry point described below. Use of either of these methods keeps CMM informed of the consequential changes in daba.

In addition to the load-overlay entry point, a more direct call, set-daba, is provided. This call is used to either remove some portion of the dynamic area from CMM control, or to add some now-unused portion of the static area to the dynamic area. One use might be to unload, in a sense, an overlay or part of an overlay. The segment loader resident uses this set-daba call to inform CMM of segment loads.

The loader computes hha when doing basic loading, overlay generation, or segment generation. The value hha remains constant under ordinary circumstances, but can change when extended overlay loading is done. The changing value of hha presents a particular problem for CMM because fixed-position blocks can only be safely positioned starting at hha and running upwards.

CMM provides a set-hha entry for the purpose of accommodating extended overlay loading; however, certain restrictions apply. First, hha can be lowered anytime, but it can only be raised when there are no active fixed-position blocks. Second, hha must always be larger than the last word address of any overlay loaded via the load-overlay entry. Thus, hha must be raised, if necessary, before doing extended overlay loading.

In some circumstances, it is desirable to switch from a CMM environment to a non-CMM environment. A deactivate-cmm entry point is provided for this purpose. CMM preserves its memory of the state of the dynamic area and its control over this area across any sequence of overlay or segmentation loads, even across loads of new (0,0) overlays; therefore, the use of this entry point is the only way for the user to remove the restrictions implied by this control.

LOAD OVERLAY

A call to this subroutine initiates the loading of an overlay according to information contained in a parameter area, the address of which is given in the call.

The logical form of the call is:

load-overlay(paddr)

paddr	The address of the parameter area. The parameter area is the same as that used by the LOADREQ macro when the CMM parameter is absent, and is described in the CYBER Loader reference manual and in the SCOPE 2 Loader reference manual.

The FORTRAN form of the call is:

CALL CMMLDV(name,l1,l2,n,u,v,e,lwa,fwa,ovlname, eptname,ne,fe,status,eptaddr)

These parameters are described in the CYBER Loader Reference Manual, section 6, and in the SCOPE 2 Loader Reference Manual, section 4.

CMM assures that the contents of all active blocks are unaffected by the loading operation. Following overlay loading, daba is reset just past the loaded overlay, which is minfl from the loaded overlay plus a delta value. Delta is zero if there is no parameter area fwa; otherwise, it is the parameter area fwa value minus the fwa from the overlay. In addition, if a (0,0) main overlay is loaded, then hha is reset to the value from the loaded overlay. At the time of this call, hha must be greater than or equal to lwa+1 of the overlay loaded. If the lwa field in the parameter area is set to a value greater than hha, CMM sets it equal to hha. Control returns to the user following the completion of overlay loading. The setting of the e bit in the second parameter word is honored; if set, control is transferred to the loaded overlay rather than to the caller. Also, the restriction of the lwa parameter is honored in that an error results if the overlay does not fit in the allotted space.

The entire procedure followed by this subroutine is to set daba up to hha, load the overlay, then reset daba back to the new minfl because the new value is not known until after the overlay is loaded. Only type 54 table overlays can be loaded by this call. Binary overlays formed before the implementation of type 54 tables cannot be loaded; if attempted, an error message is produced. Upon call completion, information is returned in the parameter area.

The COMPASS calling conventions are:

Entry point name:

CMM.LDV (entry is effected by return jump)

Entry conditions:

Register X1 contains the parameter area address, paddr

Exit conditions:

The parameter area is set with the specified returns

Registers preserved:

A0, X0, X2, X3, X4, X5, X7, B1 through B7

LOAD OVERLAY VIA FOL

A call to this subroutine indicates the loading of a higher level overlay created to be loaded by the Fast Overlay Loading feature. For more information on this feature, see the CYBER Loader reference manual. Fast Overlay Loading is not available under SCOPE 2.

The logical form of this call is:

load-overlay(ident)returning fwa and epadr

The FORTRAN form of the call is:

CALL CMMLOV(ident,fwa,epadr)

ident	The overlay name and/or level
fwa	FWA of 54 table of overlay
epadr	Entry point address of overlay

CMM assures that the contents of all active blocks are unaffected by the loading operation. Daba is reset just past the loaded overlay. There is no automatic transfer capability to the overlay when using CMM.LOV. Only higher-level (non(0,0)) overlays can be loaded using CMM.LOV.

The COMPASS calling conventions are:

Entry point name:

CMM.LOV(entry is effected by return jump)

Entry conditions are shown in table 2-5.

Exit conditions are shown in table 2-6.

Registers preserved:

A0, X0, B2, B3, X5

TABLE 2-5. CMM.LOV ENTRY CONDITIONS

Register	Bits	Contents
X1	59-18	Overlay name
	17-12	Zero
	11-6	Primary overlay level
	5-0	Secondary overlay level (the level fields are used if and only if the overlay name field is zero; otherwise, the load is by overlay name)

TABLE 2-6. CMM.LOV EXIT CONDITIONS

Register	Bits	Contents
B1	17-0	1
B6	17-0	fwa(less than zero if an error occurred; see CYBER Loader reference manual for exact values)
B7	17-0	epadr(less than zero if an error occurred)

RESPECIFY THE DYNAMIC AREA

This function permits an explicit respecification of daba, which can be either an increase or decrease.

The logical form of the call is:

set-daba(new-daba)

The FORTRAN form of the call is:

CALL CMMSDA(new-daba)

new-daba — The new dynamic area base address, which must be less than or equal to the current value of hha. If the new-daba is greater than the value of daba at the time of the call, no type 1 block group can be active.

The current value of daba is set to new-daba. (Refer to the low-memory communication words that contain daba information, discussed later in this section.) CMM assures that the contents of all active blocks are unaffected.

The COMPASS calling conventions are:

Entry point name:

CMM.SDA (entry is effected by return jump)

Entry conditions:

Register X1 contains new-daba

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

RESPECIFY THE HIGHEST HIGH ADDRESS

A call to this function provides for extended overlay loading; however, the call can be useful for purposes other than overlay loading. For example, if a program enters a phase wherein it is known that the longest overlay will never be loaded again, hha can be lowered to give CMM more space for fixed-position blocks.

The logical form of the call is:

set-hha(new-hha)

The FORTRAN form of the call is:

CALL CMMSHA(new-hha)

new-hha — The new highest high address; must be greater than or equal to the current value of daba. If greater than the current value of hha, there can be no active fixed-position blocks.

The COMPASS calling conventions are:

Entry point name:

CMM.SHA (entry is effected by return jump)

Entry conditions:

Register X1 contains new-hha

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

DEACTIVATE CMM

This call forces CMM to become inactive and to cease exercising any control over the dynamic area. From this point on, none of the programming restrictions imposed by CMM apply; the job continues as if CMM were not present. If CYBER Record Manager (CRM) is being used, a call to CMM.KIL can cause various aborts. CMM.KIL should be used with caution. Any legal first calls to CMM (listed at the end of this section) cause CMM to be reinitialized.

The logical form of the call is:

deactivate-cmm.

The FORTRAN form of the call is:

CALL CMMKIL

This call does not cause any of the content of the active blocks to change; thus, the user can continue to reference any blocks that were active at the time of this call. However, management of the contents and size of the dynamic area becomes the user's responsibility.

The COMPASS calling conventions are:

Entry point name:

CMM.KIL (entry is effected by return jump)

Entry conditions:

None

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

BLOCK GROUP CALLS

Blocks are grouped by chaining them together, beginning with a 1-word entry in the group-id table. (The format of this entry is given in section 1.) Fixed block chain pointers are in word 1 of the block header, which is present only if the block is a member of a group.

Actions related to block groups enable the user to activate a new group and to release a block group from the chain.

ACTIVATE A BLOCK GROUP

A new logical block group is activated and a unique group identification is created by CMM through this entry point. The group remains active until a call is made to free it. During the period in which a block group is active, blocks can be allocated as members of the group. Blocks that are allocated as members of a type 1 group must have a lifetime shorter than the next increase in the size of the static area, and are preferentially allocated below hha. Two things cause an increase in the size of the static area: a set-daba call, which sets a larger daba than currently exists; and a load-overlay call issued when daba is less than hha.

The logical form of the call is:

activate-group(group-type),returning group-id

The FORTRAN form of the call is:

CALL CMMAGR(group-type,group-id)

group-type — A 1-bit code that defines the type of the activated group:

- 0 Unrestricted lifetime
- 1 Lifetime limited to the next increase in daba

group-id — An integer that uniquely identifies a group of blocks.

The COMPASS calling conventions are:

Entry point name:

CMM.AGR (entry is effected by return jump)

Entry conditions:

Register X1 contains group-type

Exit conditions:

Register X2 contains group-id
Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

FREE A BLOCK GROUP

All blocks allocated as members of an identified group are freed by this call, and the group is deactivated.

The logical form of the call is:

free-group(group-id)

The FORTRAN form of the call is:

CALL CMMFGR(group-id)

group-id — An integer that uniquely identifies a group of blocks to CMM. The group must be active.

The COMPASS calling conventions are:

Entry point name:

CMM.FGR (entry is effected by return jump)

Entry conditions:

Register X1 contains group-id

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

INFORMATION CALLS

There are several calls available for obtaining useful information from CMM, ranging from the properties of a particular block to the overall picture of the state of the dynamic area.

CMM uses a number of words of the dynamic area for its own purposes in managing the dynamic area. These overhead words, while comprising a reasonably small percentage of the dynamic area under normal circumstances, are present in an amount that is a function of the number and characteristics of the active blocks. Furthermore, the number of words is in no way an external specification of CMM, and cannot remain stable between versions of CMM. These facts must be kept in mind when interpreting some of the information returned by the calls.

GET BLOCK INFORMATION

A call to this entry point obtains the current size-code, group-id, group-type, and the current block-size of a selected block.

The logical form of the call is:

get-block-info(block-fwa)returning size-code, group-type, group-id, block-size.

The FORTRAN form of the call is:

CALL CMMGBI(block-fwa,size-code,group-type, group-id,block-size)

block-fwa — The first word address of the block for which current information is desired.

Refer to the activate-group call for definition of group-type and group-id, and to the allocate-fixed call for definition of size-code and block-size.

The COMPASS calling conventions are:

Entry point name:

CMM.GBI (entry is effected by return jump)

Entry conditions:

Register X1 contains block-fwa

Exit conditions are shown in table 2-7.

Registers preserved:

A0, X0, B2, B3, X5

TABLE 2-7. CMM.GBI EXIT CONDITIONS

Register	Bits	Contents
X2	59-18	zero
	17-0	block-size
X3	59-30	zero
	29	group-type
	28-12	group-id
	11-6	size-code
	5-0	reserved for future use
X4	59-36	zero
	35-0	reserved for future use
B1	17-0	one

GET MAXIMUM AVAILABLE FIXED BLOCK SIZE

The dynamic area is examined and the size of the largest fixed-position block that can be allocated at the moment is returned. Nothing is actually allocated by this call; the value returned is valid only until the next call to any CMM entry point except this one or get-block-address.

Note that if a block is allocated with the same size as is returned by this function, the user must take care that free space is available before trying to allocate additional blocks. Attempts to allocate additional blocks are made not only by the user program, but also by overlay loads, capsule loads, and so forth.

The logical form of the call is:

get-fixed-size(fl,gt)returning size

The FORTRAN form of the call is:

CALL CMMGFS(fl,gt,size)

fl — Field length flag:

0= Size returned assumes that no fl increase is allowed

1= Size returned would cause fl to equal maxfl

gt — Group type flag:

- 0= Size returned is that of a group type 0 block, which cannot extend below hha
- 1= Size returned is that of a group type 1 block, which can extend below hha

size — A value indicating the largest fixed block that can be allocated under the conditions specified by the fl and gt flag parameters in this call.

The COMPASS calling conventions are:

Entry point name:

CMM.GFS (entry is effected by return jump)

Entry conditions:

Register X1 contains fl
Register X2 contains gt

Exit conditions:

Register X6 contains size
Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

GET STATISTICS FOR OVERFLOW RECOVERY

Through use of this call, information is returned to a designated area.

The logical form of the call is:

get-overflow-statistics, returning return-area

The FORTRAN form of the call is:

CALL CMMGOS(return-area)

return-area — Address of fixed-size area into which the returned information is placed by CMM.

Statistics concerning the current state of the dynamic area are collected and placed into the return area. The statistics returned are placed as shown in figure 2-2. Note that all values returned are integers except the current utilization level (word 4) which is a floating-point number; in FORTRAN word 4 is type REAL, in COBOL word 4 is a COMP-2 item.

The COMPASS calling conventions are:

Entry point name:

CMM.GOS (entry is effected by return jump)

Entry conditions:

None

Exit conditions:

Register X1 contains the fwa of the return area
Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

GET STATISTICS FOR JOB SUMMARY

A set of statistics concerning the overall history of the dynamic area is collected and placed into a designated return area. The statistics provide information appropriate to printing job step summaries.

The logical form of the call is:

get-summary-statistics,returning return-area

The FORTRAN form of the call is:

CALL CMMGSS(return-area)

return-area — Address of a fixed-size area into which the information is returned by CMM.

Although the returned information is of use at the end of a job step, it is also useful for measuring CMM performance. This entry point can be called anytime such statistical information is needed.

The COMPASS calling conventions are:

Entry point name:

CMM.GSS (entry is effected by return jump)

Entry conditions:

None

Exit conditions:

Register X1 contains the fwa of an area containing the following information:

Word	
0	Maximum number of allocated words
1	Maximum field length attained
2	Reserved for future use
3	Number of field length increases
4	Number of field length decreases
5	Reserved for future use

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

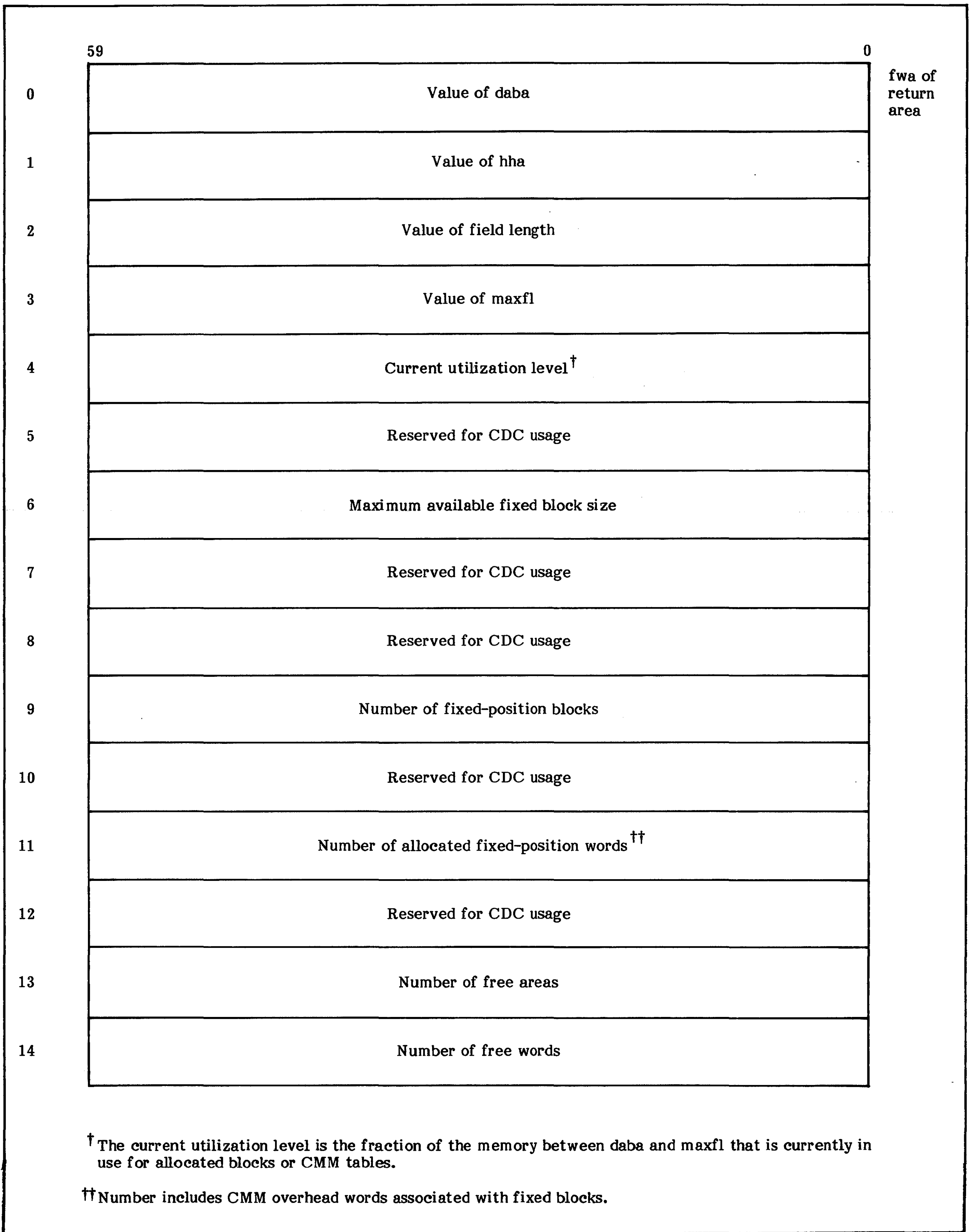


Figure 2-2. Dynamic Area Statistics

EFFICIENCY-INCREASING CALLS

Calls to the efficiency-increasing entries of CMM do not affect the actual external behavior of CMM. If properly used, these calls improve the time-performance of CMM; if improperly used, they degrade it.

OPTIMIZATION FUNCTIONS

Three separate optimization functions are available. Two of them are related to the stopping and starting of a no-readjustment period intended to help CMM save overhead processing needed for the reduction of the field length. The third informs CMM of an impending period of CMM inactivity so that the dynamic area and field length can be reduced to a minimum value.

The only logical form of the call is a return jump to one of the three CMM entry points; there are no entry conditions.

The FORTRAN form of the call is:

CALL CMMOPn

The COMPASS calling conventions are:

Entry point name (entry is effected by return jump):

CMM.OP1
CMM.OP2
CMM.OP4

Entry conditions:

None

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

CMM.OP1 Function

This function signals the beginning of a no-readjustment period. The call informs CMM that the user intends to give several consecutive calls that free-up space; these calls might be followed immediately by requests for space. Following notification by this call, CMM ceases most of its normal activity of readjusting the current field length.

CMM.OP2 Function

This function signals the end of a no-readjustment period. It informs CMM that the sequence of freeing requests is now over and that CMM is to resume its normal activity.

CMM.OP4 Function

This function causes CMM to compact the dynamic area. CMM is informed that the user intends to begin a long period of no CMM activity. CMM responds by compacting the dynamic area, which reduces the field length to eliminate any free space beyond the end of the highest fixed block.

SET-OWN-CODE ERROR PROCESSING

A call to this function sets CMM in a mode of operation that causes control to pass to a user-supplied routine only at a time when an abort would otherwise occur. The abort that could otherwise occur for the default utilization level being exceeded does not occur.

The logical form of the call is:

set-own-code(addr)

The FORTRAN form of the call is:

CALL CMMOWN(addr)

addr Address of the user error exit.

When an error is encountered, transfer of control to the specified address takes place with register B1 set to 1 and register X1 set as shown in figure 2-3. Transfer is performed through an unconditional jump rather than a return jump.

Whenever an error exit occurs in this mode, CMM restores itself to normal error processing mode. A subsequent error results in an abort unless another call to this function has been made prior to the subsequent error occurrence.

If the FORTRAN call is used, addr must be the name of an error exit routine which will be called on a CMM error. This routine must have one formal parameter in the form of an integer array with three elements. The first word will receive me, the second word ue, and the third calladr; me, ue, and calladr are defined in figure 2-3.

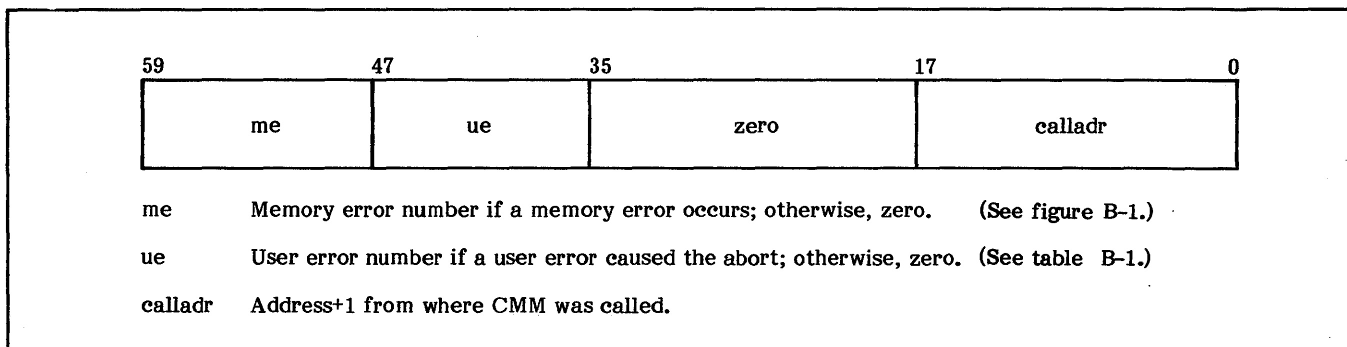


Figure 2-3. Register X1 Contents

The COMPASS calling conventions are:

Entry point name:

CMM.OWN (entry is effected by return jump)

Entry conditions:

Register X1 contains the address of the user error exit

Exit conditions:

Register B1 contains 1

Registers preserved:

A0, X0, B2, B3, X5

LOW MEMORY COMMUNICATION WORDS

Two words in low memory are used to communicate information between CMM and the loader and the job step advancer.

Bits 0 through 17 of word 65_8 are initially set to minfl by the loader or by the job step advancer when performing the first part of the initial load. When CMM first becomes active, it takes this value as the initial daba, then complements it to produce a -daba, which indicates the active state of CMM. CMM is defined to be active from the point during a job step at which it is first called, until either the termination of the job step or a call to the deactivate-cmm entry point. The value of daba remains negative as long as CMM is active, during which time these bits cannot be changed by any other program. If daba is to be altered, the set-daba function must be used. User programs utilizing CMM cannot modify these bits once CMM becomes active, and must also take care with any modifications that occur before CMM becomes active.

Bits 0 through 17 of word 104_8 contain hha, which is initially set upon completion of a relocatable load, or when the main (0,0) overlay or a root segment is loaded. This value must not be subsequently changed by any user.

The CMM interface routine CMMACT can be called via CALL CMMACT(ind) to determine if CMM is active. CMMACT tests ra+65_8; if CMM is active, ind is set to 1 on return, if CMM is not active, ind is set to 0.

CMM USE

When CMM is assembled for use in a version of an operating system, it is tailor-made to the requirements of the installation. To this end, a set of assembly options are provided, which reside in the Update common deck called CMMCOM. The options are called by most of the modules of CMM during assembly.

CMM SELECTION

There are two versions of CMM: a version that does no error checking, known as the fast version, and a version with error checking, known as the error-checking version. Both versions have identical external specifications. By use of an assembly option, one version is designated as the default version and all calls to CMM functions result in the use of the default version. To select the other version, a single LIBLOAD control statement in the load sequence for the job step is used. This method of selection can be used on normal relocatable loads only.

If the fast version is the default, but the error-checking version is desired, use:

LIBLOAD(SYSLIB,CMMSAFE)

If the error-checking version is the default, but the fast version is desired, use:

LIBLOAD(SYSLIB,CMMFAST)

LEGAL FIRST CALLS

The following functions are legal first calls to CMM. Use of any other functions produces an error message.

CMM.AGR	activate-group
CMM.ALF	allocate-fixed
CMM.FAF	flexible-allocate-fixed
FMM.FWA	get-block-fwa
CMM.GOS	get-overflow-statistics
CMM.GSS	get-summary-statistics
CMM.KIL	deactivate-CMM
CMM.LDV	load-overlay
CMM.LOV	load overlay via FOL
CMM.OP1	optimization-1
CMM.OP2	optimization-2
CMM.OP4	optimization-4
CMM.OWN	set-own-code-error-processing
CMM.SDA	set-daba
CMM.SHA	set-hha
CMM.SIV	save-identified-value

Common Memory Manager (CMM) routines can be called from one of several computer languages other than COMPASS. In using such an interface, the calling language must use the FORTRAN calling sequence conventions given in section 2. All parameters must be 60-bit integer items aligned on word boundaries. Use of languages other than COMPASS or FORTRAN requires that the user be aware of how the language assigns arrays.

Under NOS and NOS/BE, all of the language callable routines reside in the library SYMLIB; therefore, the CYBER loader directive LDSET(LIB=SYMLIB) should be included in the load sequence of all programs referring to these routines. Under SCOPE 2, these routines reside in SYMIO; the SCOPE 2 loader directive LDSET(LIB=SYMIO) should be used.

FORTRAN VERSION 5 INTERFACE TO CMM

In FORTRAN 5 programs, CMM always manages field length except when the STATIC option is specified in the FTN5 control statement. (See the FORTRAN 5 reference manual for a description of the STATIC option.) If the STATIC option is not specified, CMM ensures that the field length is increased or decreased properly to accommodate assigned blocks.

The FORTRAN 5 user can use CMM to assign blocks of memory for arrays. This assignment is completely dynamic, and the blocks should be freed when the program is finished with them.

The examples in figures 3-1 and 3-2 illustrate the use of CMM in FORTRAN programs. In both examples, CMMALF is called to allocate a fixed-position block with a length of 10 words, the value 1.0 is stored in the fifth word of the block, and the block is freed by calling CMMFRF.

In figure 3-1, the main program is used for allocating and freeing the block, and the subroutine is used to manipulate data within the block. The main program calculates an offset IOFF by subtracting the location of CMMAR(1) from IFWA. The first word of the block is given by CMMAR(IOFF+1), which is passed to the subroutine. Within the subroutine, the first element of array CMMBLK is the first word of the block. Words in the block are referenced as elements of the array, with no complicated subscript calculations necessary. For programs that manipulate matrices or other multi-dimensional arrays, CMMBLK can be defined as a multi-dimensional array.

In figure 3-2, both the CMM calls and the data manipulations are contained in the main program. This method is somewhat more prone to errors than the method shown in figure 3-1, because the offset must be included each time a word in the block is referenced, and because use of a one-dimensional array is necessary.

Note that in these examples, the arrays are of type real. For integer arrays, the same executable statements can be used. For double-precision arrays, the program must allow for each array element being two words long. For character data, the program must allow for the length of each element and for the fact that the address of the array must be masked off of the result returned by LOCF.

Refer to section 2 for descriptions of the CMM routines and their calling sequences.

All CMM interface routines for NOS and NOS/BE are in the library SYMLIB. For any FORTRAN run using the CMM interface routines, the statement LDSET(LIB=SYMLIB) must be included in the loader directives or the statement CALL SYMLIB must be included in the FORTRAN program. SCOPE 2 users must specify SYMIO instead of SYMLIB.

```
      PROGRAM CMM2
      DIMENSION CMMAR(1)
      CALL SYMLIB
      ILEN = 10
      CALL CMMALF(ILEN,0,0,IFWA)
      IOFF = IFWA-LOCF(CMMAR(1))
      CALL TWIDDLE(CMMAR(IOFF+1),ILEN)
      CALL CMMFRF(IFWA)
      END


      SUBROUTINE TWIDDLE(CMMBLK,ILEN)
      DIMENSION CMMBLK(ILEN)
        .
        .
        .
      CMMBLK(5) = 1.0
        .
        .
        .
      RETURN
      END
```

Figure 3-1. FORTRAN/CMM Example Referencing the Block as an Array

```
      PROGRAM CMM1
      DIMENSION CMMAR(1)
      CALL SYMLIB
      ILEN = 10
      CALL CMMALF(ILEN,0,0,IFWA)
      IOFF = IFWA-LOCF(CMMAR(1))
        .
        .
        .
      CMMAR(IOFF+5) = 1.0
        .
        .
        .
      CALL CMMFRF(IFWA)
      END
```

Figure 3-2. FORTRAN/CMM Example Referencing With an Offset

SYMPL INTERFACE TO CMM

In processing CMM blocks with SYMPL, the basic rule is that the block should always be a based array. In order to allocate the block, code similar to that shown in figure 3-3 should be used.

```
BASED ARRAY ABC (0:30) S(1);
  ITEM ABCITEM I(0,0,60);
        .
        .
        .
XREF PROC CMMALF I;
XREF PROC CMMFRF I;
        .
        .
        .
CMMALF(30, 0, 0, P<ABC>); #ALLOCATE BLOCK #
        .
        .
        .
ABCITEM (3) = 33; # REFERENCE AN ITEM IN BLOCK #
        .
        .
        .
CMMFRF(P<ABC>); # FREE THE BLOCK< #
```

Figure 3-3. SYMPL/CMM Interface Example

All other calls are self-explanatory. Wherever block-fwa is used, the P function of the array should be used. Sending parameters can be constants, as shown in figure 3-3. The return-area parameter in CMMGOS and CMMGSS (refer to section 2) should be the name of an array of integer items (except the fifth item in CMMGOS, which is real). The address of the user error exit (addr) for CMMOWN can be the name of a procedure (PROC) with one formal parameter that is an array of 3 integer words.

COBOL VERSION 5 INTERFACE TO CMM

COBOL programs execute CMM routines by using the ENTER statement. The subprogram name and parameters used in the ENTER statement are the same as those used in the FORTRAN call. To interface with COBOL, all parameters used in the ENTER statements should be described as COMP-1 items. In order to allocate a block, the COBOL statements shown in figure 3-4 should be used.

Note that the block size is in words, not characters. All of the size parameters in the CMM calls are in words. Some other rules are:

None of the parameters can be constants.

The routines CMMSDA, CMMSHA, and CMMKIL should not be called since they can cause interference with normal COBOL processing.

The return-area parameter in CMMGOS and CMMGSS (refer to section 2) should be group items with subordinate COMP-1 items (except for the fifth one in CMMGOS, which should be COMP-2). There should be one item for each expected return.

```
DATA DIVISION
      .
      .
      .
01  BLOCK-SIZE PIC 99 COMP-1 VALUE 50.
01  ZERO-PARAM PIC 9 COMP-1 VALUE 0.
01  BLOCK-FWA PIC 9(6) COMP-1.
PROCEDURE DIVISION
      .
      .
      .
    ENTER "CMMALF" USING BLOCK-SIZE, ZERO-
    PARAM, ZERO-PARAM, BLOCK-FWA.
      .
      .
      .
    ENTER "CMMFRF" USING BLOCK-FWA.
```

Figure 3-4. COBOL/CMM Interface Example

CMMOWN cannot be called from COBOL since there is no way to pass COBOL the address of the program; if CMMOWN is called from COMPASS, the name should be that of a COBOL subroutine with one parameter that is a group item with 3 subordinate COMP-1 items.

COBOL provides the program C.CMMMV to move data in and out of CMM blocks. C.CMMMV can perform the following actions:

Move data from a fixed-position block to the Data Division.

Move data from the Data Division to a fixed-position block.

Move data from one fixed-position block to another fixed-position block.

The following statement enters C.CMMMV:

ENTER "C.CMMMV" USING send-loc, rcv-loc, num

send-loc
: A data name specifying the sending location from which the data is moved. If send-loc is a COMP-1 item, it contains the address of the sending item (usually a fixed-position block). If send-loc is not a COMP-1 item, it contains the data being sent.

rcv-loc
: A data name specifying the receiving location to which the data is moved. If rcv-loc is a COMP-1 item, it contains the address of the receiving item (usually a fixed-position block). If rcv-loc is not a COMP-1 item, it receives the data directly.

num
: An optional data name specifying either the number of characters to be moved or the beginning of an area within the fixed-position block. In both cases, the default for num is 1.

: When both send-loc and rcv-loc are COMP-1 items (that is block-fwa's), num is the number of characters to

be moved. When either send-loc or rcv-loc is not COMP-1, num acts as a subscript into the fixed-position block. The block is divided into areas that are the size of the non-COMP-1 item; num identifies which area within the block from or to which data is to be moved. The first area is numbered 1.

Figure 3-5 shows a COBOL program that illustrates the use of CMM with COBOL. In the program, a 50-word fixed-position block is allocated; the block is filled with 10 words of A's, 10 words of spaces, 10 words of C's, 10 words of spaces, and 10 words of E's, in that order. Next, a second, 100-word fixed-position block is allocated and the contents of the first block are moved to the last half of the second block. The last half of the second block is then moved back to the interface area where it is tested to see if it contains the original data (10 words of A's, 10 words of spaces, 10 words of C's, 10 words of spaces, and 10 words of E's). Finally, statistics concerning the dynamic area are printed, both blocks are freed, and statistics concerning the dynamic area are printed again.

In the figure, notice the following points about C.CMMMV:

In line 100, when data is moved from INTERFACE-AREA to the first block, the num parameter indicates into which 10 words of the block the data is moved. The size of INTERFACE-AREA determines how many words are moved.

In line 76, when data is moved from the first block to the second block, the num parameter indicates how many characters are moved.

In line 79, when data is moved from the second block to INP-AREA, the num parameter has the default value of 1; the first 50 words beginning at the address given by TEMP are moved. The size of INP-AREA determines how many words are moved.

```
COBOL Program:

 1              IDENTIFICATION DIVISION.
 2              PROGRAM-ID.  CMMIF.
 3              DATA DIVISION.
 4              WORKING-STORAGE SECTION.
 5              01  BLOCK-FWA                PIC 9(6) COMP-1.
 6              01  BLOCK2-FWA               PIC 9(6) COMP-1.
 7              01  BLOCK-SIZE               PIC 9(6) COMP-1.
 8              01  CMM-INDEX                PIC 9(5) COMP-1.
 9              01  GROUP-ID                 PIC 9(6) COMP-1.
10              01  INP-AREA                 PIC X(500).
11              01  R-OFFSET                 PIC 9(6) COMP-1.
12              01  TEMP                     PIC 9(6) COMP-1.
13              01  ZERO-PARAM               PIC 9(6) COMP-1 VALUE 0.
14              01  CHECK-AREA.
15                  02  FILLER               PIC X(100) VALUE ALL "A".
16                  02  FILLER               PIC X(100) VALUE SPACES.
17                  02  FILLER               PIC X(100) VALUE ALL "C".
18                  02  FILLER               PIC X(100) VALUE SPACES.
19                  02  FILLER               PIC X(100) VALUE ALL "E".
20              01  GOS-AREA.
21                  02  DABA                 PIC 9(6) COMP-1.
22                  02  HHA                  PIC 9(6) COMP-1.
23                  02  FL                   PIC 9(6) COMP-1.
24                  02  MAXFL                PIC 9(6) COMP-1.
25                  02  UL                            COMP-2.
26                  02  FILLER               PIC 9(3) COMP-1.
27                  02  MAX-SIZE             PIC 9(3) COMP-1.
28                  02  FILLER               PIC 9(6) COMP-1.
29                  02  FILLER               PIC 9(6) COMP-1.
30                  02  FPBLK                PIC 9(6) COMP-1.
31                  02  FILLER               PIC 9(6) COMP-1.
32                  02  ALLW                 PIC 9(6) COMP-1.
33                  02  FILLER               PIC 9(6) COMP-1.
34                  02  FRAR                 PIC 9(6) COMP-1.
35                  02  FWDS                 PIC 9(6) COMP-1.
36              01  INTERFACE-AREA.
37                  02  IF-DATA              PIC X(10) OCCURS 10 TIMES.
38
39              PROCEDURE DIVISION.
40              STARTT.
41                  DISPLAY "STARTING CMM INTERFACE TEST".
42            *   ALLOCATE A CMM BLOCK (50 WORDS)
43                  MOVE 50 TO BLOCK-SIZE.
44                  ENTER "CMMALF" USING BLOCK-SIZE, ZERO-PARAM, ZERO-PARAM,
45                    BLOCK-FWA.
46                  DISPLAY "FIRST BLOCK ALLOCATED".
47            *   FILL THE BLOCK WITH SPACES
48                  MOVE SPACES TO INTERFACE-AREA.
49                  PERFORM MOVE-TO-BLOCK VARYING CMM-INDEX FROM 1 BY 1
50                    UNTIL CMM-INDEX > 5.
51                  DISPLAY "FIRST BLOCK SPACE FILLED".
52            *   SET THE THIRD OCCURRENCE OF THE INTERFACE AREA IN THE CMM
53            *   BLOCK TO ALL "C".
54                  MOVE ALL "C" TO INTERFACE-AREA.
55                  MOVE 3 TO CMM-INDEX.
56                  PERFORM MOVE-TO-BLOCK.
57            *   SET FIRST AND LAST TO A AND E
58                  MOVE ALL "A" TO INTERFACE-AREA.
59                  MOVE 1 TO CMM-INDEX.
60                  PERFORM MOVE-TO-BLOCK.
61                  MOVE ALL "E" TO INTERFACE-AREA.
62                  MOVE 5 TO CMM-INDEX.
63                  PERFORM MOVE-TO-BLOCK.
64                  DISPLAY "FIRST BLOCK INITIALIZED WITH A, C, AND E".
65            *   ALLOCATE A SECOND BLOCK (100 WORDS)
66                  MOVE 100 TO BLOCK-SIZE.
67                  ENTER "CMMALF" USING BLOCK-SIZE, ZERO-PARAM, ZERO-PARAM,
68                    BLOCK2-FWA.
```

Figure 3-5. Sample COBOL/CMM Program (Sheet 1 of 2)

```
 69                DISPLAY "SECOND BLOCK ALLOCATED".
 70           *   MOVE FIRST BLOCK TO LAST PART OF SECOND
 71           *   NOTE THAT THE RECEIVING BLOCK IS BEING OFFEST BY ADDING 50
 72           *   WORDS (NOT CHARACTERS) TO ITS FWA
 73                ADD 50 BLOCK2-FWA GIVING TEMP.
 74           *   LENGTH IS IN CHARACTERS
 75                MOVE 500 TO BLOCK-SIZE.
 76                ENTER "C.CMMMV" USING BLOCK-FWA, TEMP, BLOCK-SIZE.
 77                DISPLAY "FIRST MOVED TO SECOND".
 78           *   NOW MOVE LAST PART OF SECOND TO WS AND CHECK IT
 79                ENTER "C.CMMMV" USING TEMP, INP-AREA.
 80                IF CHECK-AREA = INP-AREA
 81                    DISPLAY "DATA TESTS OUT CORRECTLY"
 82                ELSE
 83                    DISPLAY "TEST FAILED - WRONG DATA READ"
 84                    DISPLAY " SHOULD BE ", CHECK-AREA
 85                    DISPLAY "        IS ", INP-AREA
 86                .
 87           *   GET THE STATS ON USAGE AT THIS TIME
 88                ENTER "CMMGOS" USING GOS-AREA.
 89           *   DISPLAY THE STATS IN OCTAL
 90                ENTER "C.DSPDN" USING GOS-AREA.
 91           *   RETURN THE BLOCKS
 92                ENTER "CMMFRF" USING BLOCK-FWA.
 93                ENTER "CMMFRF" USING BLOCK2-FWA.
 94                ENTER "CMMGOS" USING GOS-AREA.
 95                ENTER "C.DSPDN" USING GOS-AREA.
 96                DISPLAY "TEST COMPLETED".
 97                STOP RUN.
 98
 99            MOVE-TO-BLOCK.
100                ENTER "C.CMMMV" USING INTERFACE-AREA, BLOCK-FWA, CMM-INDEX.
```

Execution of the program:

```
STARTING CMM INTERFACE TEST
FIRST BLOCK ALLOCATED
FIRST BLOCK SPACE FILLED
FIRST BLOCK INITIALIZED WITH A, C, AND E
SECOND BLOCK ALLOCATED
FIRST MOVED TO SECOND
DATA TESTS OUT CORRECTLY
 $$$L=0090  000000000000000106310000000000000001063100000000000000013200
000000000000003777001710570501741712554717177463146314631463
            : : : : : : : A F Y : : : : : : : A F Y : : : : : : : A Z :
: : : : : : : 4 ; : O H . E A @ 0 J   * 0 0 @ % L % L % L %
 $$$        000000000000003645000000000000000000010000000000000000000000
000000000000000000060000000000000000000000000000000000001246
            : : : : : : : 3 + : : : : : : : : : : A : : : : : : : : : :
: : : : : : : : : F : : : : : : : : : : : : : : : : : : : : J -
 $$$        000000000000000000000000000000000000000300000000000000365520

            : : : : : : : : : : : : : : : : : : : : C : : : : : : : 3   P

 $$$L=0095  000000000000000106310000000000000001063100000000000000013200
000000000000003777001710451704416444505261717746314631463
            : : : : : : : A F Y : : : : : : : A F Y : : : : : : : A Z :
: : : : : : : 4 ; : O H + 0 D 6 " + E V 0 0 @ % L % L % L %
 $$$        000000000000003645000000000000000000010000000000000000000000
000000000000000000040000000000000000000000000000000000001016
            : : : : : : : 3 + : : : : : : : : : : A : : : : : : : : : :
: : : : : : : : : D : : : : : : : : : : : : : : : : : : : : H N
 $$$        000000000000000000000000000000000000000300000000000000365750

            : : : : : : : : : : : : : : : : : : : : C : : : : : : : 3 . /

TEST COMPLETED
```

Figure 3-5. Sample COBOL/CMM Program (Sheet 2 of 2)

STANDARD CHARACTER SETS A

CONTROL DATA operating systems offer the following variations of a basic character set:

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE or SCOPE 2, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described above for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table (table A-1) are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC Graphic	CDC Hollerith Punch (026)	CDC External BCD Code	ASCII Graphic Subset	ASCII Punch (029)	ASCII Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	%††	8-6	16	%††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	↱	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0	52	!	12-8-7	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-0	72	<	12-8-4	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	^ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end-of-record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

DIAGNOSTICS

Two types of errors occur during CMM use: memory errors and user errors. Memory errors occur when insufficient memory is available for a CMM call. User errors occur when a CMM call is made incorrectly.

When the fast version of CMM is used, only memory errors are detected. When the error-checking version is used, both memory and user errors are detected.

MEMORY ERRORS

When CMM aborts because of insufficient memory, the message shown in figure B-1 is sent to the dayfile. This message indicates that growth or allocation of a block was requested or that more space was needed for CMM internal tables, but not enough additional space was available.

When a memory error occurs, either more central memory space must be provided or the space used must be reduced. The amount of central memory provided is given by the value of maxfl. This value is set in the following places: the CM parameter of the job statement, the MFL control statement on NOS, or the access limits at login time.

When the error-checking version of CMM is used, an additional message shown in figure B-2 is sent to the dayfile. This message appears in the dayfile before the message shown in figure B-1. The value me in the message indicates the specific condition that caused the error; me is also contained in register X1 in the format shown in section 2.

USER ERRORS

User errors are detected only when the error-checking version of CMM is used. The messages shown in table B-1 are sent to the dayfile when a user error occurs. The number identified as ue in the column following the error message is not sent to the dayfile, though it is associated with the message; ue indicates the user error number contained in register X1 in the format shown in section 2.

```
CMM CENTRAL MEMORY LIMIT INSUFFICIENT - MAXFL IS nnnnnnB.

nnnnnn      octal value of maxfl.
```

Figure B-1. Memory Error Message

CMM ERROR me, MAIN-CMMxxx, OV-CMMyyy

me — The specific error condition, designated by a numeric value having the following meaning:

1. The default utilization level, as set by assembly option DEFTRIG, has been reached.
2. The default utilization level has not been reached, but the request, which is probably for a large amount of space, cannot be fulfilled.
3. The field length available to CMM is not long enough to accommodate internal CMM tables. The maximum field length permitted for the job should be increased if possible.
4. Same as 2 above, except error occurred in overflow mode; overflow is currently used only by PL/I.
5. The dynamic area is so full that CMM cannot add space to one of its internal blocks (this is not likely to occur because the internal blocks generally increase by only a few words at a time. For this to occur, the field length would be at maxfl).

xxx — Name of the function currently in execution.

yyy — Related to the overflow mode; currently, yyy is expressed as three dashes (- - -) if not in overflow mode.

Figure B-2. Memory Error Message in Error-Checking Version

TABLE B-1. USER ERROR MESSAGES

Message	ue	Significance	Action
BLOCK FWA ERROR	4	Functions that specify a block-fwa must specify the fwa of an active block.	User error. Correct and resubmit.
BLOCK POINTERS MESSED UP - SEE (B2)	35	One of the CMM internal words in the dynamic area has been destroyed. The address of the destroyed word is in register B2.	System error. Consult system analyst.
BLOCK SIZE ERROR ON ALLOCATE	5	On block allocate functions, the size must be greater than or equal to zero.	Correct blocksize on allocate function.
FIXED-POS, LWA-END GROWTH BLOCK PRESENT	8	If a fixed-position, lwa-end growth block exists, then no other fixed blocks can be allocated while this block remains active.	User error. Correct fixed-block allocation.
FWA OR LWA GT HHA	9	For CMM.LDV, both the specified and resulting fwa and lwa must be less than hha.	Correct fwa and/or lwa. Resubmit.
GROUP-TYPE 1 BLOCKS ILLEGALLY ACTIVE	11	If daba is increased due to a CMM.LDV or CMM.SDA request, no type 1 group can be active. This is always the case for CMM.LDV unless the highest overlay was in at the time of the call, meaning daba was equal to hha.	User error. Correct and resubmit.
ILLEGAL 1ST CALL TO CMM	12	Many of the functions, due to their nature, must not be the first function called (for example, CMM.GLF, CMM.FGR, etc.).	See list of legal first calls at the end of section 2.
ILLEGAL GROW-SHRINK AMOUNT	13	On a grow request, the value must be positive or zero. On a shrink request, the value must be positive or zero, but must not be greater than the current block size.	Correct grow-shrink amount.
IMPROPERLY SPECIFIED GROUP-ID	15	For CMM.FGR, the group identified by group-id must be active.	Correct group-id to active.
IMPROPERLY SPECIFIED GROUP-TYPE	16	For CMM.AGR, group-type must be 0 or 1.	Change group-type on CMM.AGR to 0 or 1.
IMPROPERLY SPECIFIED SIZE-CODE	17	For fixed blocks, size-code is a 3-bit value.	Change size-code to a 3-bit value.
MAXFL LT HHA+100, CMM CANNOT FUNCTION	21	MAXFL must be larger than HHA (length of the longest overlay, or, for nonoverlaid jobs, the length of the program plus the library routines) plus 100_8 words.	Increase MAXFL (maximum field length). On batch jobs, NOS or NOS/BE, use CM parameter on job card. On NOS, increase FL for associated user name, or (if MFL statement is used), increase number in MFL statement. If NOS or NOS/BE system default is too small, reduce required FL for this job.
MAY NOT ADD LWA GROWTH TO FIXED BLOCK	22	For CMM.CSF, lwa-end growth cannot be added to a fixed block; i.e., size-code cannot be specified as xx1 unless it is already xx1.	User error. Correct and resubmit.
MAY NOT INCREASE DABA GT HHA	24	For CMM.SDA, new-daba must be less than or equal to the current value of hha.	Correct daba to less than or equal to hha.
MAY NOT REDUCE HHA LT DABA	25	For CMM.SHA, new-hha must be greater than or equal to the current value of daba.	Correct hha to be greater than or equal to daba.

TABLE B-1. USER ERROR MESSAGES (Contd)

Message	ue	Significance	Action
MAY NOT UP HHA IF FIXED BLOCKS PRESENT	26	For CMM.SHA, if new-hha is greater than the current value of hha, there can be no active fixed blocks.	User error. Correct and resubmit.
NON-ACTIVE GROUP-ID SPECIFIED	28	If group-id is nonzero on allocate or free-group functions, it must identify an active group.	Correct group-id to active.
NON-EXISTENT UNLOAD-ID	30	For CMM.SUA, the unload-id parameter must be an unload-id value previously returned by CMM.SUA.	Correct unload-id to value previously returned by CMM.SUA.
NON-54 TABLE OVERLAY LOADED	31	Only type 54 table overlays can be loaded via CMM.LDV.	Correct using table 54 overlays.
OVERLAY LOAD INCREASED HHA	32	The hha cannot be increased via CMM.LDV; a CMM.SHA function must be issued first.	User error. Correct and resubmit.
RA+65B INCORRECT - ICM CALLED FROM nnnnnn	36	Lower 18 bits of ra+65_8 contain complement of daba. This address or word at daba is incorrect because the address must be complemented the first time CMM is called; or the address is complemented again after CMM.KIL is called and bit 17 must be zero; or word pointed to by address (should be daba) does not contain FL, thus address is incorrect or daba has been destroyed; or address is not complemented when CMM is active, indicating address has been destroyed (bit 17 is zero). Value nnnnnn contains address of CMM routine last called by user.	Determine cause of destruction of ra+65_8 and correct the error.
SIZE-CODE VIOLATION	34	The respective bit must be set in the size-code to allow the specified growth or shrinkage.	Correct size-code with respective bit.

GLOSSARY C

This appendix contains definitions for terms unique to the Common Memory Manager. Included are terms frequently used with regard to Control Data computer systems.

Always-resident Routine -
A routine referenced by all of the CMM functions required to be continually in memory.

CMM Functions -
A CMM routine that performs a specific user-requested task.

Dynamic Area -
An area in memory which can vary in size according to predetermined rules.

Dynamic Space -
Areas in central memory and/or storage devices that are in a state of continuous change or involved in productive activity.

Field Length -
The area in central memory allocated to a particular job; the only part of central memory that a job can directly access.

Group -
A set of data, which can include words, blocks, or other items. A separator, such as a header, defines the logical boundary between groups.

Header -
A word or sequence of words that contain information related to the data in words that follow a header.

Interface Routine -
A routine that links the operating system with two or more programs that can be members of the product set. A routine shared by two or more computer programs.

Job Step -
The processing associated with a single control statement.

Job Step Approach -
The portion of the operating system that initiates control statement processing.

Lifetime -
An attribute related to type 1 block groups in CMM. Before an increase in the static area can occur, all block groups of type 1 must be freed. Therefore, all type 1 block groups have a lifetime shorter than the time between successive increases in the static area size.

Logical Group -
A collection of groups independent of their physical environment.

Offset -
A value which, when applied to a base or relative address, produces an absolute address.

Pointer -
A field or word containing an address, either direct or indirect, and information related to the item to which it points. Usually points to the first word address of an item, but can point to the last word occupied.

Product -
A software program produced and/or supplied by the computer manufacturer and intended for use in conjunction with the operating system.

Product Set -
A group of software programs, including the operating system, which control and supplement the execution of computer programs.

Stack -
Portions of computer memory or registers used for temporary storage. To assign a position or order, as to stack jobs for batch processing.

Static Area -
An area in memory established during initial loading that remains fixed in size between loads.

Status -
A state or condition.

Subprogram -
A part of a larger program; can be converted into machine instructions independently.

Task -
A piece of work assigned to a component part of the operating system and to be completed within a certain period of time.

Trailer -
A word or sequence of words that contain information related to the data in words that precede the trailer. A trailer record follows a group of records and contains pertinent data related to that group of records.

Tree -
The relationship of components to one another, such as files or program segments, the diagram of which is in the form of a tree and depicts branching from an original point. Each branch on the tree has a definable parent node, which is the representation of a state or an event by means of a point in the diagram.

Utilization Level -
That percentage of memory not available to CMM for allocation. Refer to DEFTRIG assembly option in text.

INDEX

Activate a block group 2-8
activate-group subroutine 2-8
allocate-fixed-returning subroutine 2-1
Allocate or create fixed-position blocks 2-1

Backward pointer 1-4
Basic loading 1-2
block-fwa parameter 2-1
block-size parameter 2-1
Blocks and block groups
    Activate a block group 2-8
    Block group pointers 1-5
    Calls 2-8
    Definition 1-4
    Fixed block header 1-5
    Freeing blocks 1-4, 2-8
    Word format 1-5
Bootstrap code 1-6

Change fixed-position blocks 2-4
Change-specs-fixed subroutine 2-4
Change static area size 1-6
CMM
    Control 1-6
    Deactivate 2-8
    Field length required 1-2
    Internal storage 1-6
    Operation and use 2-1
    Selection 2-13
    Use 2-12
COBOL interface to CMM 3-2

daba 1-1, 1-3
Deactivate CMM 2-8
Dynamic area
    Base address pointer word format 1-3
    Description 1-3
    Header and trailer word formats 1-4
    Statistics 2-9

Efficiency-increasing calls 2-12
Entry point names
    CMM.AGR 2-8
    CMM.ALF 2-1
    CMM.CSF 2-4
    CMM.FAF 2-2
    CMM.FGR 2-8
    CMM.FRF 2-3
    CMM.FWA 2-5
    CMM.GBI 2-9
    CMM.GFS 2-9
    CMM.GLF 2-4
    CMM.GOS 2-10
    CMM.GSS 2-10
    CMM.KIL 2-8
    CMM.LDV 2-6
    CMM.LOV 2-6
    CMM.OP1 2-12
    CMM.OP2 2-12
    CMM.OP4 2-12
    CMM.OWN 2-12
    CMM.SDA 2-7

Entry point names (Contd)
    CMM.SFF 2-3
    CMM.SHA 2-7
    CMM.SIV 2-5
    CMM.SLF 2-3
Error-checking version 2-13
Example
    COBOL 3-2
    FORTRAN 3-1
    SYMPL 3-2

Fast version 2-13
Fixed block header 1-5
Fixed-position block calls 2-1
Fixed-position blocks
    Adjustments 2-1
    Allocate or create blocks 2-1
    Free or destroy 2-3
fl parameter 2-9
flexible allocate 2-2
FORTRAN calls
    CMMAGR 2-8
    CMMALF 2-1
    CMMCSF 2-4
    CMMFAF 2-2
    CMMFGR 2-8
    CMMFRF 2-3
    CMMFWA 2-5
    CMMGBI 2-9
    CMMGFS 2-9
    CMMGLF 2-4
    CMMGOS 2-10
    CMMGSS 2-10
    CMMKIL 2-8
    CMMLDV 2-6
    CMMLOV 2-6
    CMMOPn 2-12
    CMMOWN 2-12
    CMMSDA 2-7
    CMMSFF 2-3
    CMMSHA 2-7
    CMMSIV 2-5
    CMMSLF 2-3
FORTRAN interface to CMM 3-1
Free a block group 2-8
free-fixed subroutine 2-3
free-group subroutine 2-8
Free space header 1-6

Get block address 2-5
get-block-fwa subroutine 2-5
get-block-info subroutine 2-9
Get block information 2-9
get-fixed-size subroutine 2-9
Get maximum available fixed block size 2-9
get-overflow-statistics subroutine 2-10
Get statistics for job summary 2-10
Get statistics for overflow recovery 2-10
get-summary-statistics subroutine 2-10
Group-id entry 1-4
group-id parameter 2-1
group-type parameter 2-8
grow-at-lwa-fixed subroutine 2-4
Grow fixed-position blocks 2-4

Highest high address (hha) 1-1

id parameter 2-5
Illustrated CMM terminology 1-2
Information calls 2-9
Initial loading 1-2

Language-callable routines 2-1
Legal first calls 2-13
Libraries
    SYMIO 2-1
    SYMLIB 2-1
load-overlay subroutine 2-6
Load-overlay via FOL 2-6
Low memory communication words 2-12

Maximum field length value (maxfl) 1-3
minfl 1-3

new-daba parameter 2-7
new-hha parameter 2-7
new-size-code parameter 2-4
num parameter 2-3, 2-4

Optimization functions 2-12
Overlay loading 1-2
Overlays 2-5

paddr parameter 2-6

Register X1 contents 2-12
Respecify the dynamic area 2-7
Respecify the highest high address 2-7
return-area parameter 2-10

Save identified value 2-5
Segment loading 1-2
Segments 2-5
set-daba subroutine 2-7
set-hha subroutine 2-7
Set-own-code error processing 2-12
Shrink at fwa 2-3
Shrink at lwa 2-3
Shrink fixed-position blocks 2-3
size-code parameter 2-1
size parameter 2-10
Static area 1-2
SYMPL interface to CMM 3-1

unique-id parameter 2-5
User/CMM control 1-6

value parameter 2-5

COMMENT SHEET

MANUAL TITLE: Common Memory Manager Version 1 Reference Manual

PUBLICATION NO.: 60499200 REVISION: F

NAME:

COMPANY:

STREET ADDRESS:

CITY: STATE: ZIP CODE:

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

___ Please reply ___ No reply necessary

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE TAPE

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P.O. BOX 3492
Sunnyvale, California 94088-3492

CUT ALONG LINE

FOLD FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION