File No. S360-31
GY28-2018-3

Program Logic

Version 8.1

IBM System/360 Time Sharing System

System Service Routines

Describes the internal logic of the system service routine used in TSS/360. The system service routines are non-resident programs that can be invoked either directly by the user, through the use of system commands and macro instructions, or indirectly, in response to requests from other system components.

The system service routines are divided into eight categories: catalog services; external storage allocation; device management; virtual memory allocation; small virtual memory allocation; symbolic library services; control section store; and serviceability aids. Appendixes describe VAM and SAM DSCB formats and catalog SBLOCK formats.

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

Prerequisite Publications

The reader must be familiar with the information presented in:

IBM System/360 Time Sharing System: System Programmer's Guide, GC28-2008
IBM System/360 Time Sharing System: System Logic Summary PLM, GY28-2009

PREFACE

This book describes the internal logic of the system service routines used in TSS/360. The system service routines are divided into eight categories and each category is discussed in a separate section of this book:

1. Catalog Services
2. External Storage Allocation
3. Device Management
4. Virtual Memory Allocation
5. Small Virtual Memory Allocation
6. Symbolic Library Services
7. Control Section Store
8. Serviceability Aids

Each of the 8 sections provides a general description of the catagory and also individual routine descriptions. Section 9 contains the flowcharts for all the routines. Appendixes include a module directory and descriptions of VAM and SAM DSCB formats and catalog SBLOCK formats.

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

The reader must be familiar with the information presented in:

IBM System/360 Time Sharing System: System Programmers Guide, GC28-2008

IBM System/360 Time Sharing System: System Logic Summary PLM, GY28-2009

Fourth Edition (September 1971)

This is a major revision of, and makes obsolete, GY28-2018-2 and Technical Newsletters GN28-3124 and GN28-3152. Changes since the latest Technical Newsletter include four new Catalog Service routines -- USERCAT SCAN (CZUFY), CATFLUSH (CZCFX), DSCB/CAT RECOVERY (CZUFX), Catalog Error Processor (CZCFE), and a new External Storage Allocation routine, READWRIT (CZCEM). SHAREUP (CZCFU) and LOCATE (CZCFL) have been changed extensively and smaller changes have been made to INDEX (CZCFI) and MOUNTVOL (CZCAM).

This edition applies to Version 8, Modification 1, of the IBM System/360 Time Sharing System, and to all subsequent releases until otherwise indicated in new editions or Techincal Newsletters. Before using this publication, please refer to the latest edition of IBM System/360 Time Sharing System: Addendum, GC28-2043, which lists the current editions of publications.

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

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Dept. 643, Neighborhood Road, Kingston, New York 12401

CONTENTS

IBM System/360 Time Sharing System, hereafter referred to in this publication as Time Sharing System/360 (TSS/360), provides eight categories of system service routines. These routines are invoked either directly by the user through use of system commands and macro instructions, or indirectly, in response to requests from other system components.

The eight categories are:

- Catalog Services: These routines allow the user to read, write, update, and share his user catalog and the data sets contained in it. The catalog is a hierarchial index structure; each index level is made up of 64-byte blocks of storage called SBLOCKs. Catalog service routines operate on the index levels and the SBLOCKs.

- External Storage Allocation: These routines control the storage on direct access volumes used for data storage. One group of routines controls the initial allocation of storage to a data set and the secondary allocation of storage as it is needed. When storage is no longer needed, it can be returned in part, or all the space assigned to the data set can be returned when the data set is deleted. One set of ESA routines is used for the processing of SAM format volumes and another for VAM format.

- Device Management: These routines are used to allocate, mount, and release private devices. Mounting messages are issued to the operator; and his reply is awaited. Facilities are provided to mount subsequent volumes of a multivolume physical sequential data set on the same device as earlier volumes of the same data set.

- Virtual Memory Allocation: This routine provides for problem program requests for dynamic allocation and release of virtual storage during program execution. In addition, an existing block of storage can be dynamically expanded. The storage obtained can be either private or shared.

- Small Virtual Memory Allocation: This routine provides the system or user program with dynamic allocation of virtual storage by bytes. SVMA obtains integral numbers of pages from VMA which are then segmented to satisfy the requests.

- Symbolic Library Service Routines: A symbolic library is composed of two portions: an index and a symbolic component. Each portion may be a distinct data set, or they may be members of different partitioned data sets. The symbolic portion contains source statements; the index portion is used to locate the desired group of symbolic statements. Routines are provided to automatically index and retrieve individual sections from the library.

- Control Section Store Routine: This routine is invoked by the CSTORE macro instruction. It enables the user to create, during program execution, a control section that is placed as a module in the current JOBLIB.

- Serviceability Aids: These aids consist of service programs to record and retrieve statistical data concerning system performance and hardware failures, and a routine which converts time and date from system format to EBCDIC form.

GENERAL DESCRIPTION OF CATALOG SERVICES

THE CATALOG

The system catalog (called the scratch catalog) is a VAM partitioned data set that exists in virtual storage from startup to shutdown. This is a dynamic catalog since only members (users) that are active during a session exist in the scratch catalog. The individual user catalogs are VS data sets with format U records that reside on public storage.

When a user logs on, the scratch catalog is searched for the user's member, and normal logon procedure follows if it is found. If the user's member is not found, the external VS member will be copied into the scratch catalog. Subsequent references will use the scratch catalog.

When the user logs off (or abends), a check is made to see if the scratch catalog member has been changed. If it has, the system copies the scratch catalog member into the external VS data set. At shutdown, the scratch catalog is erased. If a scratch catalog exists at startup (no shutdown), it will be used as the system catalog. Entries in the catalog contain infomation about the physical location of data, a list of users who have access to the data, and how the data may be accessed. The catalog is a hierarchical structure of indexes, each uniquely identifiable by its symbolic name plus the symbolic name of each higher-level index in its structure. The highest level of index in the catalog contains one entry for each authorized user of the system. Each entry is the eight-character userid, which is concatenated automatically by the system to the name that the user assigns to his data set. This highest level index is called the master index, and is actually the partitioned organization directory (POD) of the catalog data set.

Each index name is referred to as a simple name; combining the names produces a qualified name. If the name of each level, from highest to lowest, is specified, a single data set is identified by its fully qualified name. If one or more of the lowest levels are not included in the name, a collection of data sets is identified by its partially qualified name. Including all simple names and separating periods (but excluding the user identification), the length of a data set name must not exceed 35 characters. The concatenation of the user identification gives the name a maximum length of 44 characters for catalog references.

A data set is cataloged at OPEN time, using information placed in the JFCB by the DDEF command. As pages are assigned to members, they are formatted into 64-byte blocks called SBLOCKs which are the basic unit of information in the catalog. Each SBLOCK contains indexing information and pointers to related SBLOCKs. Space for each user catalog is allocated in whole page units, 64 available SBLOCKs per page. Related SBLOCKs are chained together in groups called indexes, each of which correspond to a level of qualification in the data set name structure adopted by the user.

A logical entity within the catalog consists of one or more chained SBLOCKs. The logical entities defined in an SBLOCK are:

- Indexes
- Generation indexes
- Data set descriptors
- Sharing descriptors
- Sharer lists

As pages are assigned to user catalogs, the fields are zeroed and information is inserted in groups of 64 bytes. Unused SBLOCKs are identified by a control field of binary zeros. The form of a typical SBLOCK is shown in Figure 1. (Refer to Appendix B for a detailed description of SBLOCKs.)

The SBLOCKs within an index need not be contiquous. Forward and backward links in each SBLOCK give the relative location of the next succeeding and preceding SBLOCKs in the index level, which may be in the same or different pages. Within each index are SBLOCKs containing the name of each subordinate index, and a pointer to the beginning of the SBLOCK chain of each subordinate index. For a fully qualified data set name in SAM, the lowest level index in the catalog of the data set owner contains one or more SBLOCKs identifying the volume or volumes on which the data set resides. In VAM if the data set is public, the SBLOCK points to the format E DSCB and gives the volume type. If private, it points to the format E DSCB and the volume

FIELD	#BYTES	DESCRIPTION
1	3	(CCCFWD) Forward Pointer to the first character of the next SBLOCK in the chain. Pointer is of the form Pbb: P is the logical page number within the member; bb is the relative byte within the page.
2	1	(CCCCT1) Binary Count of SBLOCKs allocated from a page. This field is maintained by Catalog Services in the first SBLOCK of each page.
3	3	(CCCBWD) Backward pointer to the preceding SBLOCK in a chain. Pointer is of the form Pbb.
4	1	(CCCCT2) Binary count of bytes allocated from the field to follow.
5	56	Allocatable field; format is variable according to SBLOCK usage.

Figure 1. Typical SBLOCK Format

on which it resides. Such SBLOCKs are called data set descriptors. The lowest level index of a sharer's fully qualified name (FQN) in the catalog of a data set sharer consists instead of an SBLOCK containing the owner's FQN for the data set, which may in turn be used to locate the SBLOCK in the owner's catalog that identifies the volume. Such an SBLOCK in the sharer's catalog is called a sharing descriptor.

In addition to pointers to subordinate indexes and other user catalogs, SBLOCKs within an index level may contain names of other users who may share data sets defined in (or subordinate to) this index level. Such SBLOCKs are called sharer lists.

If, in the course of searching through a user catalog according to a fully qualified name (FQN), the lowest level index is found to contain a sharing descriptor pointing to another owner's catalog, then the search continues in the owner's catalog using the owner's FQN. When the last index level pointed to by the owner's FQN is found, a check is made in the owner's catalog to see if the sharer is allowed to access the data set. The search for sharing information begins at the lowest level (last level found) and will continue up to the highest level (userid) if necessary, until an access for the sharer is found. If an access is not found, permission is denied the user to access the data set being searched for in the owner's catalog. The PERMIT command enables the user to add to or modify sharer lists in any index level of his catalog, and thus enables him to control the sharing privileges associated with a particular data set or group of data sets defined in his catalog, depending on which index level is affected by the command.

The catalog modification is always made to the index level corresponding to the last component of the name contained in the command. This level might consist of a single data set descriptor or sharing descriptor, or if this level is sharable, it might indicate the presence of subordinate levels, which are sharable as a unit.

The virtual partitioned access method is used to load the user catalog into a user's virtual storage. When a user enters a LOGON command into the system, a constant job file control block (JFCB) for the catalog is entered into his virtual storage. This JFCB points to the format E DSCB for the catalog and indicates that the catalog is a shared data set. Once the user identification is established, virtual memory task initialization (VMTI) is used to open the catalog data set; this causes the RESTBL and POD to be brought into the user's virtual storage. A subsequent FIND and GET bring the member corresponding to the user's catalog into virtual storage. Any of the various catalog services to be performed on the member are performed by a catalog service routine. A member appears in virtual storage as a contiguous piece of data. Addresses used within a member are relative to an origin address of zero. Catalog service routines use the origin address of the member as an index so that the members are effectively address-free and can be located in any contiguous area of virtual storage.

The STOW macro instruction can be used to update the POD of the catalog to indicate compression or expansion for new or deleted members. If a page is required to expand a member in virtual storage, the FINDEXPG routine adds on a contiguous page. The external address is assigned through the PUT macro instruction. Access method routines will assign an external page from extents already allocated to the catalog if there is a page available; if not, the routines will request another extent from external storage allocation. If a user should QUIT the system, the DELETE routine updates the POD and returns the external storage used by that user's catalog for allocation to other members.

CATALOG PROTECTION

Any portion of storage containing the catalog is assigned a storage protection key different from that of the user data and programs. In addition, another form of protection is required because catalog services may be executed in parallel; this protection is called an interlock.

Parallel catalog accesses may be data dependent (i.e., they may modify the same data, causing interference if the modification is concurrent). In order to avoid interference problems, an interlock is placed in the first index level of each user catalog. The interlock operates in such a manner that a user catalog can be accessed only in a serial manner. A catalog service routine will force a time-slice end if it finds itself locked out of a member.

CATALOG SERVICES ROUTINES

Time Sharing System/360 contains catalog service routines designed to allow the user to update, add to, and delete from his private catalog. Catalog service routines are reenterable, closed service routines residing in virtual storage. They operate only in the privileged state and are enterable only by a privileged program. The privileged program calls the catalog service routine, supplying a list of input parameters.

Catalog service routines are divided into two groups:

- Routines invoked by the user program.
- Routines invoked by other catalog service routines.

Routines Invoked by the User Program

The following catalog service routines can be invoked by the user from his terminal by means of commands. The routines are privileged, however, and cannot be called directly by the user's program.

ADDCAT (CZCFA)
: is invoked at OPEN time. This routine establishes the required new entries in the index levels of the user catalog.

DELCAT (CZCFD)
: is invoked when a user issues a DELETE command, or a CATALOG command renaming a data set. This routine removes the lowest level qualifier(s) of the old data set name from the user catalog.

SHARE (CZCFS)
: is invoked when a data set owner issues a PERMIT command extending sharing privileges. SHARE creates within the specified index level of the owner's catalog a list of sharers and their privilege codes.

UNSHARE (CZCFV)
: is invoked when a data set owner issues a PERMIT command rescinding sharing privileges. This routine reverses the operation performed by the SHARE routine by removing names from a sharer list in the owner's catalog or by restricting the access privileges of some sharers.

SHAREUP (CZCFU)
: is invoked when a user other than the data set owner issues a SHARE command. SHAREUP places a reference to the owner's catalog (the owner's FQN) into the sharer's catalog (under the sharer's FQN).

LOCATE (CZCFL)
: is invoked to locate the index level in the catalog that corresponds to the lowest level qualifier in a fully or partially qualified data set name. LOCATE does this by searching through each index level corresponding to the components of the name. At the same time, if a sharing descriptor is encountered during the search, the owner's FQN is prefixed to the remaining components of the user's FQN, and at the last stage of the search the owner's lowest index level is checked for being sharable by the user. Every catalog service routine except GETSBLOCK and SEARCHSBLOCK calls upon LOCATE to find a path leading to particular index levels within the user catalog. In response to one user command, LOCATE may thus be called many times.

INDEX (CZCFI)
: is invoked to make a catalog entry for a new name during the processing of a CATALOG or SHARE command. This routine directs a search for existing index levels corresponding to the first components of the name, and then constructs new index levels for the remaining qualifiers for which no index levels previously existed.

CATALOG ERROR PROCESSOR (CZCFE)
: is invoked to execute all SYSER and ABEND macro instructions currently claimed by the catalog service routines. Whenever one of these routines encounters a user input data format error and no appropriate return codes are available, CZCFE is called to

invoke a completion code 1 ABEND. CZCFE will also be called to execute a SYSER and ABEND when one of the routines discovers a catalog structure error. In this case, CZCFE will also write a message to SYSLOG describing the type and location of the error in the catalog.

Routines Invoked by Other Catalog Service Routines

The following additional catalog service routines are not invoked directly by the user, but are called by other catalog service routines in order to carry out frequently performed operations upon the catalog.

GETSBLOCK (CZCFG)
As the need arises to process various SBLOCKs within the catalog, this routine finds each SBLOCK by means of its page and byte displacement relative to the catalog member, reads the page containing the SBLOCK into storage if necessary, and gives its virtual address to the caller.

SEARCHSBLOCK (CZCFH)
As empty SBLOCKs are required to expand or create new catalog entries, this routine locates an unused 64-byte block within the catalog member and also fills in the forward and backward links to chain the new empty SBLOCK into an existing index.

Routines Invoked by Other System Routines for Dynamic Catalog Operation

The following routines perform the dynamic catalog function, which consists of bringing individual user catalogs into a scratch catalog as users log on the system. The scratch catalog is then used as a system catalog.

USERCAT SCAN (CZUFY)
If the system finds that the DSCB for the SYSSVCT data set is in error, this routine is called to rebuild the SYSSVCT data set.

CATFLUSH (CZCFX)
This routine is used to copy members into the user catalog and delete members from the scratch catalog.

DSCB/CAT RECOVERY (CZUFX)
This routine is used to rebuild a user catalog if the current member in the scratch catalog cannot be used. Also, this routine is used to rebuild a user catalog if no member exists in the scratch catalog, and the user catalog is not usable. If the user catalog is rebuilt, all sharing information must be reentered by the user because it is lost when the user catalog is rebuild from public DSCBs.

ADDCAT (CZCFA)

ADDCAT is a reentrant, privileged routine residing in virtual storage. This routine adds new data sets to the catalog, updates the catalog for new data sets, and deletes outmoded generations as required. (See Chart AA.)

Entry Points:
CZCFA1 - for SAM data sets.
CZCFA2 - for VAM data sets.

Input: ADDCAT has two separate sets of input parameters. Register 1 contains the address of a parameter list utilized as follows:

For SAM data sets (CZCFA1)

Word 1 Address of a 44-byte FQN
Word 2 Address of a halfword option code

byte 1 = 00 Normal processing
byte 1 = 80 Update requested
byte 2 = 00 Do not update user field
byte 2 = 80 Update user field

Word 3 Address of 26-byte user field
Word 4 Address of a packed parameter word

bits 0-7 = 00 Read only user priv.
= 01 Read/write user priv.

bits 8-15 label data
bits 16-23 data set origin
bits 24-31 volume count

Word 5 Address of the start of volume serial numbers

For VAM data sets (CZCFA2)

Word 1 Address of JFCB

Word 2 Address of fullword option code

Word 3 Address of 64-byte return area

Word 4 Address of fullword pointer to public/private volume table.

The option code functions are as shown in Figure 2. No entry implies the bit is unused.

Output: The catalog is updated.

Assumptions: A new data set is not a generation data set unless so indicated in the SBLOCK.

Bit Number	Bit=0	Bit=1
0	No update	Update
1-3		
4		RET
5		CATVAM
6	DSD update	JFCB update
7	Complete update	Update specific fields
8-10		
11	No change	Share privileges
12		RET and access privileges
13		Label data
14		Data set organization
15		DSCB/CATALOG RECOVERY
16-18		
19		Tape density
20		Tape parity
21		
22		DSCB pointer
23-30		
31		Device type code

Figure 2. Option Codes for VAM Input Parameter

If the maximum number of generations has been reached and the 'delete oldest only' indicator is not on, all generations shall be deleted from the catalog and erased.

Modules Called:
INDEX (CZCFI) -- To create the necessary index levels for a new nongeneration catalog entry.

GETSBLOCK (CZCFG1) -- To locate an SBLOCK and calculate its VMA. (CZCFG4) -- To perform PUT/PUTX/STOW sequence as necessary.

DELCAT (CZCFD) -- To delete SAM data sets.

DELVAM (CZCFT2) -- To delete and erase VAM generation data sets.

LOCATE (CZCFL) -- To locate and retrieve SBLOCKs in the catalog to determine whether a data set is new and to provide the address of the last catalog SBLOCK and an image of the last SBLOCK located.

GATE (CZATC) -- To write messages to inform the user that a data set was not scratched when deleted or when DELVAM was unsuccessful.

SCRATCH (CZCES) -- To delete DSCBs from a SAM generation data set.

ADDDSCB (CZCEK) -- To assign space for a new format E DSCB.

SEARCH SBLOCK (CZCFH) -- To acquire and chain an empty SBLOCK for DSD or an extended SBLOCK.

FINDJFCB (CZAEB) -- To locate a JFCB for a SAM generation data set.

RELEASE (CZAFJ) -- To free a JFCB for a SAM generation data set.

ABEND (CZACP) -- To terminate processing after a system error.

CATALOG ERROR PROCESSOR (CZCFE) -- To execute SYSERs and ABENDs when claimed by ADCAT or when a catalog structure error is discovered.

Exits:
Normal - Register 15 contains 00.

Error - Register 15 contains one of the following codes:

SAM DATA SETS
04 - Userid not found in POD
08 - FQN not found
0C - No sharing allowed

VAM DATA SETS
08 - FQN already in catalog

Operation: ADDCAT is entered with a Type I linkage and has two entry points; one for VAM data sets and one for SAM data sets.

A. SAM Data Sets:

ADDCAT checks for generation and nongeneration data sets.

1. Nongeneration: For nongeneration data sets, Type I linkage is used to enter INDEX to create the necessary index levels for the FQN. ADDCAT takes one of the following actions according to the return code from INDEX:

Code '0'
indicates that the INDEX operation was completed successfully. All necessary index levels of the FQN were created, and INDEX has obtained an SBLOCK for the creation of a data set descriptor (DSD) and back chained it to the index level pointer. ADDCAT then proceeds to build a data set descriptor in the SBLOCK provided by INDEX. If the data set resides on more than one volume, additional SBLOCKs as required are assigned via the SEARCHSBLOCK program for the recording of the additional volume control fields.

Code '4'
indicates the LOCATE function was unsuccessful due to invalid USERID and

this code is passed back to ADDCAT's calling routine.

Code '8'
indicates the FQN is currently in the user's catalog. If the last level located is a DSD, the input parameters are checked to see if the user wishes to update the DSD in the catalog (attributes or vol. control fields). If no update option was indicated, processing is terminated and control is returned to the calling program. When the option of updating is taken, further checks are taken to determine the exact fields to be updated. Data must appear in the correct relative locations and if the volume control fields are being updated, a complete update of the field must be done, including the count field.

Code 'C'
indicates that no sharing is allowed of the catalog the user is trying to extend. Control is returned to the calling routine with this return code.

2. Generation: For a generation associated FQN, Type I linkage is used to enter LOCATE. ADDCAT takes one of the following actions according to the return code from LOCATE:

Code '0'
indicates that the FQN was found in the catalog implying that the user wants to perform an update. A branch is made to the code which checks and processes the update options.

Code '4'
indicates that the user's ID was not found in the POD and the code is returned to the user along with control.

Code '8'
indicates the full FQN was not found which should be the normal condition when processing generations. ADDCAT checks the last index found to determine if it is a generation index and, if not, the error code is returned to the user. If the index was a generation index, ADDCAT checks the number of generations existing against the maximum allowed. If the maximum is not exceeded, the non-generation procedure for a new data set is followed, except that a generation flag is set prior to calling INDEX and constructing the DSD.

ADCAT checks for catalog structure errors in the generation index. If an error is detected, the ERROR PROCESSOR is called (with information describing the type of error and its location in the index) to issue a SYSER and a completion code 1 ABEND.

When the maximum number of generations is exceeded, the delete options are checked to determine whether to scratch or delete all existing generations or only the oldest. If a data set is to be scratched, FINDJFCB is called to find or create a JFCB, SCRATCH is called with the JFCB returned, the JFCB is released, and DELCAT is called to remove the catalog entry. If a data set is to be deleted, FINDJFCB is called to find a JFCB, DELCAT is called to remove the catalog entry, and RELEASE is called to free the JFCB if one was found. This process is repeated until all the necessary generations have been deleted as specified at which time ADDCAT returns with a successful return code.

Code 'C'
indicates the user is not allowed to share.

B. VAM Data Sets:

LOCATE is entered to determine whether the data set is old or new, and, in the case of a generation data set, to convert the generation-version number to absolute form if it is relative. If the index levels are not all found (RC = '08'), the data set is treated as new; RC='00' indicates an old data set. Return codes of '04' and '0C' result in SYSERs (minor software) followed by ABEND.

After return from LOCATE, ADDCAT checks for the RET option in the parameter list (word 2). If the option is 1, a branch is taken to the code for updating the catalog; if the RET option is not on, ADDCAT examines the CATVAM option to add a private VAM data set to the catalog without going to ADDDSCB (since the data set already has a DSCB). If the option is on for an old data set, control is returned to the user with a code of '08' indication that the fully qualified name given by CATVAM was not unique. The DSD is also returned to the user.

If the CATVAM option is on for a new data set ADDCAT checks for a generation qualifier and calls INDEX if necessary, to create a generation index.

Processing differs for new nongeneration data sets, new generation data sets, and old data sets, as follows:

1. New Nongeneration Data Sets: For a new data set it is assumed that if the last index pointer found by LOCATE is not a generation index (CCCFL1 ≠ 02), the new data set is not generation associated. Therefore, if the last qualifier is in absolute G0000V00 format, ADDCAT will return with a code of '08', except in the case of the CATVAM option being set as mentioned above. ADDDSCB is entered to obtain a DSCB for the new data set and its pointer. In the event of an error return from ADDDSCB, processing will be terminated with a SYSER (minor software) followed by ABEND. The pointer to the DSCB is inserted into the JFCB and INDEX is entered to create the necessary index levels for the FQN in the catalog. A return code of '00' from INDEX indicates an SBLOCK has been obtained for the creation of a DSD. GETSBLOCK is entered with the pointer to the SBLOCK from INDEX to get the SBLOCK; ADDCAT then gets the backward pointer to the index SBLOCK and enters GETSBLOCK to indicate in the SBLOCK the type of DSD (public or private). GETSBLOCK is entered to again get the DSD SBLOCK. (Returns of '04' or '0C' from INDEX result in SYSER and ABEND; a return of '08' indicates that the FQN was already in the catalog and results in a return to the user with a code of '08'.) ADDCAT then proceeds to construct the DSD using information in the DSD portion of the JFCB and calls CZCFG4 for PUT/PUTX/STOW. Control is returned to the user with a successful return code of '00' in general register 15 and an image of the DSD in the user's 64-byte return area.

2. New Generation Data Sets: If ADCAT is called to perform an operation on a generation data set, validity checks are performed on the generation index in the catalog. If an error is detected, the ERROR PROCESSOR is called with information describing the type and location of the error in the catalog index. The ERROR PROCESSOR issues a SYSER and completion code 1 ABEND. Otherwise ADDCAT checks the sharing access and only the owner or a sharer with unlimited access will be allowed to proceed (ABEND is otherwise invoked). GETSBLOCK is entered to get the generation index and the number of pointers to existing generations is incremented by the one to be added and compared to the maximum allowed. When a user requests deletion of generation data sets, the data sets are automatically scratched by DELVAM as well because once a data set is deleted from the catalog it is inaccessible.

If the maximum number of generations is not exceeded, ADDCAT will search through the pointers to previously existing generation data sets, comparing the new generation name to those already cataloged. The pointer to the new generation data set descriptor is sorted by generation number into the list of pointers. INDEX is entered to find and chain an SBLOCK in which to construct the DSD for the new data set.

If the maximum number of generations is exceeded, the delete options are examined to calculate the number of generations to be deleted and erased. DELVAM is entered with the FQN of the generation to be deleted. If DELVAM was unsuccessful, a message is sent to the user with the reason for no deletion and the data set name (via the GATWR macro). The number of generations to be deleted is decremented and processing continues with the next generation. DELVAM is called for each generation until the count is zero. ADDCAT then processes the new generation as above when there are no deletions to be made.

Old Data Sets: For an old data set, ADDCAT examines the option code to see if an update is requested. If not, processing is terminated with a SYSER (minor software) and an ABEND. To process the update options, ADDCAT must first determine whether the DSD or the JFCB is to be updated. (Note: Only bytes 21-55 of either the DSD or the DSD portion of the JFCB may be updated at one time. Not both.)

If the JFCB is to be updated from the DSD, the option code is examined to determine whether a complete or partial update should be performed. If a complete update is requested, ADDCAT moves each field of the DSD into the corresponding field in the JFCB. A partial update is requested, the last three bytes of the option are examined bit by bit to determine which fields are to be updated. In either case, when updating TDTAQL, the userid in Task Common is compared to the userid in the FQN to determine whether to move the owner's (CCCFL4) or sharer's (CCCFL3) access privileges into the JFCB.

If the DSD is to be updated from the JFCB, GETSBLOCK is entered to obtain the DSD SBLOCK from the catalog. The option code is examined to see if a complete or partial update is requested and the update is performed the same as the JFCB update. When the SBLOCK has been updated as required, CZCFG4 is called.

Upon completion of the updating, control is returned to the user. ADDCAT passes back an image of the DSD, and the proper return code, as indicated under 'exits', in register 15.

DELCAT (CZCFD)

DELCAT is a reenterable, nonrecursive, privileged subroutine, residing in virtual storage. It deletes index levels from the catalog structure and recatalogs index levels under a different fully qualified name. DELCAT calls LOCATE to get the specified index level and then determines if an owner's catalog is referred to by checking the first byte of the 45-byte buffer used as an entry parameter to LOCATE. If the flag is set, the sharer disposition flag in the 64-byte SBLOCK retrieval buffer is checked. (See Chart AB.)

Entry Point: CZCFD1

Input: Register 1 contains a pointer to this list.

Word 1 Pointer to fully qualified name

Word 2 Pointer to option (if RENAME option is selected, bits 0-15 contain X'04')

Word 3 Pointer to new fully qualified name (applicable only if RENAME option is selected)

Output: Register 1 contains a pointer to the input parameter list.

Modules Called:
LOCATE (CZCFL) -- To locate an index level.

INDEX (CZCFI) -- To construct chained index levels and create new members in catalog data sets.

GETSBLOCK (CZCFG1) -- To locate SBLOCK and calculate virtual storage address.
(CZCFG4) -- To perform PUT/PUTX/STOW functions.

CATALOG ERROR PROCESSOR (CZCFE) -- To claim a SYSER and a completion code 1 ABEND when a catalog structure error is encountered or an end-of-data-set is detected by GETSBLOCK.

Exits:
Normal - register 15 contains 00 - DELCAT successful.

Error - DELCAT returns one of the following codes:

04 - userid not in the POD

08 - the DELETE fully qualified name not found by LOCATE

08 - the DELETE fully qualified name is an index level containing pointers to lower levels

08 - the DELETE fully qualified name and RENAME fully qualified name refer to different user catalogs

08 - the RENAME fully qualified name is not unique

0C - indicates a sharing error

Operation: The fully qualified name supplied in the entry parameter list is used to locate the specified index level, through use of the LOCATE routine. If LOCATE returns with a non-zero return code, indicating an unsuccessful locate, DELCAT terminates with an appropriate error return code.

Upon a successful return from LOCATE, DELCAT determines if an owner's catalog was referred to by checking the first byte of the 45-byte buffer used as an entry parameter to LOCATE. If the flag is on, the sharer disposition flag in the 64-byte SBLOCK retrieval buffer is checked. In order to be able to delete from the owner's catalog, the sharer must have unlimited sharing privileges.

The fully qualified name can only describe one of the following:

- An empty index level
- A data set descriptor
- A sharing descriptor

Otherwise, DELCAT is terminated with an error indication.

DELCAT can be entered with one of the following options: DELETE or RENAME.

If the delete option is selected, the entire entity located is zeroed and the SBLOCK-count for the page is decremented by one for each SBLOCK freed. The deletion includes extended SBLOCKs and any attached sharer's lists. Next, the pointer field referred to by the back-chain in the first SBLOCK of the deleted entity, is retrieved and deleted. If the deleted pointer was the sole member of that index level, the entire index level is, in turn, deleted as described above. However, if the first SBLOCK in the user's catalog becomes empty or if the SBLOCK is a generation index, it is not deleted. If the index level is not emptied, the zeroed pointer entry is left in the index level for future use. However, if the deleted member is located in an extended SBLOCK which has become empty as a result of the deletion, the extended SBLOCK

is removed from the chain. The empty node SBLOCK is not deleted unless there are no extended SBLOCKs chained to it. DELCAT updates the respective allocated byte fields and forward and backward chains involved in the deletion and justification.

When deleting a data set descriptor from the catalog, the structure of any sharing lists or volume lists attached to it are checked for errors. If an error is encountered the ERROR PROCESSOR is called (with information describing the type and location of the error in the catalog) to execute a SYSER and a completion code 1 ABEND.

If the rename option is selected, DELCAT verifies via LOCATE that the RENAME fully qualified name and the DELETE fully qualified name refer to the same user's catalog. If they refer to different user catalogs, an error return results. DELCAT then enters INDEX to construct the new fully qualified name and transfers the contents of the last qualified level in the old fully qualified name to the new one. Once the data is successfully transferred, DELCAT proceeds to delete the last qualified level of the old fully qualified name as described above, except for deleting any sharer's list or extended SBLOCKs that are now associated with the new fully qualified name.

If the last qualified level of a fully qualified name is a sharing descriptor, the sharing descriptor is deleted, rather than the shared index level referred to.

SHARE (CZCFS)

SHARE is a reenterable, nonrecursive, privileged subroutine residing in virtual storage, that adds sharing privileges to a catalog level. An unshared level can be set to shareable, or a shared level can have its sharing access modified. Sharing can be universal (meaning that any user may share), or selective (meaning that only those users whose userid is included in the input parameter list are allowed to share). LOCATE is called to retrieve the proper level for the fully qualified name supplied. For selective sharing, a sharing list is created or updated, depending on the type of request. (See Chart AC.)

Entry Point: CZCFS1

Input: Register 1 contains a pointer to this parameter list:

Word 1 Virtual storage address of a 44-byte field containing the fully qualified name, left-justified

Word 2 Virtual storage address of a fullword control field

Word 3 Count field - number of sharers to be added to the sharers list (used only for universal sharing)

Word 4 Virtual storage address of a list of sharers

The format of the control field pointed to by Word 2 is shown in Figure 3.

Notes to Figure 3:

Byte 0:
not used.

Byte 1:
bit 0=0
indicates that a Universal/Selective sharing parameter is not present.

bit 0=1
indicates that a Universal/Selective sharing parameter is present in byte 2. This bit is examined only in the update mode. If it is 0, the sharing mode is taken from the catalog itself.

bit 1=0
access code is not present in byte 3 of the control word or in the list of new sharers. The default is unlimited access for all sharers on the list of new sharers.

bit 1=1
access code is present.

Byte 2: Universal/Selective parameter
bit 6=1
indicates that the request is for selective sharing.

bit 7=1
indicates that the request is for universal sharing. If this bit is set, the access code must be specified in byte 3.

Byte 3: access code
bits 0-7
all zero indicates unlimited sharing.

bit 1=1
indicates read/write access.

bit 2=1
indicates read-only access.

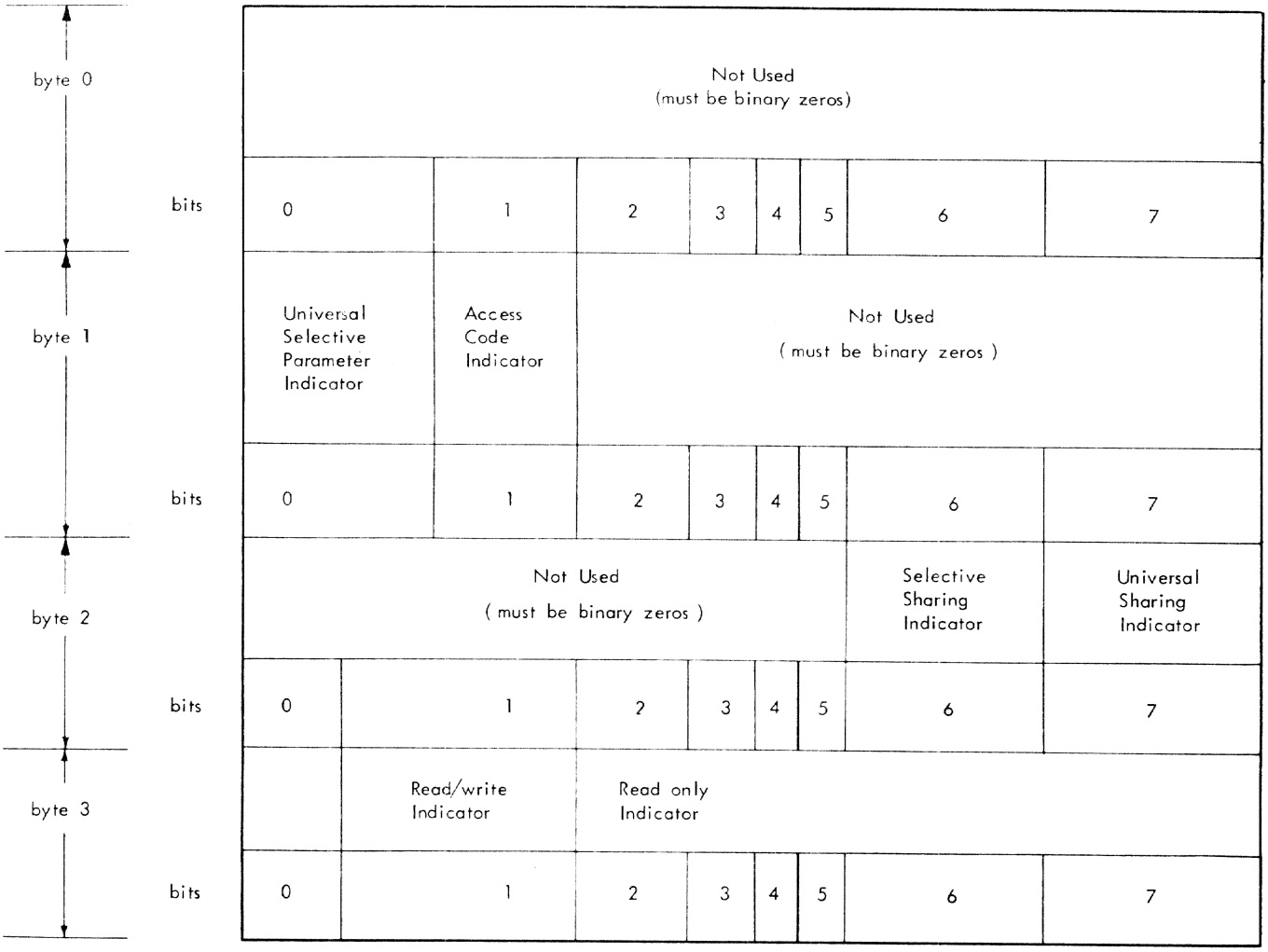


Figure 3. Format of SHARE Control field

The list of sharers pointed to by parameter 4 has a length of 9*N bytes, when N is the number of sharers indicated in Word 3 of the input parameter list. Each entry has the following format:

	SHARER ID							ACCESS CODE
Bytes	1	2	3	4	5	6	7	8	9

The access code has the same meaning as in the control field pointed to by parameter 2 (byte 3). This list is not meaningful for universal sharing.

Output: Register 1 contains a pointer to the input parameter list.

Restrictions: A user can authorize sharing only if those index levels are in his own user catalog.

Modules Called:
LOCATE (CZCFL) -- To locate an index level.

GETSBLOCK (CZCFG1) -- To locate SBLOCK and calculate virtual storage address.
(CZCFG4) -- to update external storage.

SEARCHSBLOCK (CZCFH) -- To acquire and chain an empty SBLOCK to an index level.

CATALOG ERROR PROCESSOR (CZCFE) -- To execute a SYSER and a completion code 1 ABEND when a catalog error is encountered or if an end-of-data-set is detected by GETSBLOCK.

Exits:
Normal - register 15 contains 00: SHARE successful

Error - if SHARE is unsuccessful, register 15 contains one of the following codes:

04 - Owner-id not in the POD

08 - Fully qualified name not in catalog

0C - Owner is not allowed

10 - Not used

14 - A request to share is directed to a sharing descriptor

18 - Not used

1C - Not used

20 - Request to update from a list of shares to a universal shared level or vice versa

Operation: The owner of a catalog can set a level in the catalog to "universally shareable," meaning that any user can share, or to "selectively shareable" meaning that only those users whose ID is included in the input parameter list are allowed to share.

The user can specify whether the level for a newly shared data set is to become universally shared or selectively shared.

If the level is to be universally shareable, the user can supply the access code (read-only, read/write, or unlimited access). If the user does not specify any access code, as indicated by byte 1 of the control field, unlimited access is assumed. If a sharer has unlimited access, he can add to or delete from the shared portion of the catalog. SHARE supplies LOCATE with the fully qualified name to get the proper index level. SHARE sets the sharing flag in that level to universal and sets the sharing privileges from the access code in the parameter list or will default the access to unlimited if none is given.

If the level is to be selectively shared, the user must supply a count of the number of sharers plus a list of these sharers in the form: sharer's-id (8 bytes) followed by his sharing privilege (1-byte access code). If the user does not supply any sharing privilege for the sharers on the sharing list, each sharer will be given unlimited access. This is indicated when bit 1 of byte 1 of the control field is zero. After locating and retrieving the SBLOCK associated with the fully qualified name, SHARE sets the sharing flag in the level to "selective." A sharing list is constructed and attached to the level by filling in the pointer to the sharer list. SBLOCKs to create the sharer list are obtained using SEARCHSBLOCK. CZCFG4 is used, in all cases, to update external storage when necessary.

The user can share a data set which is already being shared, but a request to change the sharing mode from selective to universal, or from universal to selective, without first restricting the data set, will not be honored and will result in a return code of 20.

If the sharing mode is universal and the user wants to leave it as universal, SHARE simply changes the sharing privileges by using the access code from the input. If none is supplied in the input parameter list, unlimited sharing privilege is assumed. When the sharing mode is selective and the user wants it to remain selective, the user must supply a count of the number of users to be added to the present sharing list plus a list of new sharers containing sharerids followed by their sharing privileges. Before updating the selective sharing, SHARE checks the structure of the sharing list for errors. If an error is encountered the ERROR PROCESSOR is called (with information describing the type and location of the error in the sharing list) to execute a SYSER and a completion code 1 ABEND. The sharing privileges for the new sharers on the sharing list can be defaulted as indicated by byte 1 of the control word. When this happens, the new sharers will be given the sharing privilege of the last sharer on the present sharing list. SHARE checks each userid to ensure that it does not already belong to the user's list. If it already belongs to the list, the access code from the input is inserted in the existing entry; if it does not already belong to the list, it is added. After the list has been completely updated, the count of sharers in the catalog level is updated by the number of IDs added.

CZCFG4 is used, in all cases, to update external storage when necessary.

UNSHARE (CZCFV)

UNSHARE is a reenterable, nonrecursive, privileged subroutine residing in virtual storage, that removes sharing privileges from a catalog level. First the proper level is located and checks are made to see that the sharing mode of the level is compatible with the type requested. If the sharing mode is universal and the request is to delete all sharers, the sharing flag is set to private and the new index level is updated in the catalog. If the sharing mode is selective and the request is to delete all sharers, the additional operation of deleting the sharer's list is performed. When the sharing mode is selective and the request is not to delete all mem-

bers, the sharer's list is searched and only the members passed in the parameter list are deleted. (See Chart AD.)

Entry Point: CZCFV1

Input: Register 1 contains a pointer to this parameter list:

Word 1 Pointer to a 44-byte field containing the fully qualified name, left-justified.

Word 2 Pointer to a fullword control field.

Word 3 Pointer to halfword count field.

Word 4 Pointer to a list of sharers.

The control field pointed to by Parameter 2 is aligned on a fullword boundary and has the following structure.

Byte 0 - unused

Byte 1 - unused

Byte 2
bits 0-5 -- unused

bit 6=1 -- delete all sharers

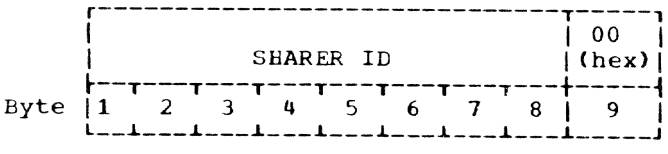
bit 7=1 -- delete only those sharers identified in the input parameter list

Byte 3 - unused

All unused bits must be set to zero.

The count field pointed to by Parameter 3 is a halfword binary count of the number of sharers to be removed from the sharer list of a selectively shared level; it is aligned on a halfword boundary. This parameter is meaningful only for selective sharing.

The list of sharers pointed to by Parameter 4 has a length of 9*N bytes, where N is the number of sharers identified in the count field above. Each entry in the list has the following format:



Output: Register 1 contains a pointer to the input parameter list.

Restrictions:

1. Sharing privilege can be removed from a catalog level only by the owner of the catalog.

2. To determine the sharing privilege of a catalog level, UNSHARE examines only the sharing privilege pointed to by the fully qualified name. This means that for UNSHARE, the sharing privilege of any higher level in the catalog has no effect on the sharing privilege of subordinate levels. The owner must specifically establish the sharing privilege of a level via SHARE if any other user is to link to the specific level via SHAREUP.

3. A level is considered private by UNSHARE until a SHARE is given for it.

Modules Called:
LOCATE (CZCFL) -- To locate an index level.

GETSBLOCK (CZCFG1) -- To locate SBLOCK and calculate virtual storage address.
(CZCFG4) -- To effect PUT/PUTX/STOW functions.

CATALOG ERROR PROCESSOR (CZCFE) -- To execute a SYSER and completion code 1 ABEND when a catalog structure error is encountered or an end-of-data-set is detected by CZCFG1.

Exits:
Normal - register 15 contains 00, UNSHARE successful

Error - register 15 contains one of these codes:

04 - Owner-id not in POD

08 - Fully qualified name not in catalog

0C - Owner is not allowed to share

10 - A list of sharers was provided for a universally shared level

14 - Request to unshare is directed to a sharing descriptor

18 - Level is not now shareable

1C - Input ids are not in the sharer's list

Operation: The owner of a catalog can use one of two options with UNSHARE:

- Making a universally or selectively shared level in the catalog private.

- Removing sharing privileges from a list of users for a selectively shared catalog level.

In either case, UNSHARE locates the fully qualified name to get the proper level in the catalog. If the level indicated is a sharing descriptor or if the level is not already shared, an error return is made.

Before removing any userids from a sharing list, the sharing list is checked for catalog structure errors. If an error is encountered, the ERROR PROCESSOR is called (with information describing the type and location of the error) to issue a SYSER and a completion code 1 ABEND.

Collective Removal

If sharing privileges are to be removed collectively, (that is, the catalog level is to be made private), UNSHARE checks the sharing mode of the level to see if it is universal. If so, it resets the sharing flag to make the level private. The pointer to the sharing list is saved and zeroed; all SBLOCKs in the sharing list are then returned to the catalog.

Selective Removal

If the sharing privileges are to be removed selectively, UNSHARE checks the sharing flag in the catalog level. If selective sharing is not indicated, an error return is made. UNSHARE next searches the sharing list for the userid. If the userid is not present, a flag is set in the access byte following the userid on the input list and UNSHARE begins to search for the next userid in the input list. If the userid is present, its entry is zeroed on the sharing list. If the deletion results in an empty sharing list SBLOCK which is not the node sharing list SBLOCK, the SBLOCK will be removed from the sharing list chain. Processing is completed when all the userids on the input list have been removed from the sharing list. If any deletion results in the removal of all of the members from the sharing list, the catalog level is set to private.

Note: For both collective and selective removal, CZCFG4 is used to update external storage when necessary.

SHAREUP (CZCFU)

SHAREUP is a reenterable, nonrecursive, privileged subroutine, residing in virtual storage. It links a user's private catalog to a level in another user's shareable catalog by constructing a sharing descriptor in the sharer's catalog that points to a node in a user's catalog that was previously designated as shareable. The shared index level is retrieved to determine if the calling program is allowed to share. If the calling program or user is allowed to share and his fully qualified name is unique, a sharing descriptor is constructed by the INDEX routine. (See Chart AE.)

Entry Point: CZCFU1

Input: Register 1 contains a pointer to the following parameter list:

Word 1 Pointer to owner's fully qualified name.

Word 2 Pointer to user's fully qualified name.

Output: Register 1 contains a pointer to the input parameter list.

Modules Called:
LOCATE (CZCFL) -- To locate an index level.

INDEX (CZCFI) -- To construct a chained index level and create new members in the catalog data set.

GETSBLOCK (CZCFG1) -- To locate an SBLOCK and calculate virtual storage address. (CZCFG4) -- To perform PUTX function.

CATALOG ERROR PROCESSOR (CZCFE) -- To execute a SYSER and completion code 1 ABEND when a catalog structure error is encountered or an end-of-data-set is detected by CZCFG1 or an unsuccessful return code is obtained from INDEX.

Exits:
Normal - register 15 contains 00; SHAREUP successful

Error - register 15 contains one of the following codes:

04 - userid not in POD

08 - user name not unique

0C - index level requested to share is not shareable

10 - owner FQN nonexistant

14 - ownerid not in POD

18 - request to add sharing descriptor with gdg.

1C - user is attempting to share another catalog in the owners catalog.

Operation: LOCATE is called to retrieve the owner's fully qualified name that is in the parameter list. If a X'04' return code is returned from LOCATE, it is converted by SHAREUP to a X'14' return code which is returned to the caller after the sharing descriptor is built. Similarly, a X'08' return code (level to be shared is not in the catalog) is converted to a X'10' return code which is sent to the caller after the sharing descriptor is built. A X'0C' return code (level cannot be shared) is sent to the caller after the sharing descriptor is built. Before processing is continued, a check is made to see if LOCATE crossed a catalog boundary. If it did, processing is terminated and the sharing descriptor is not built since sharing across multiple members is not permitted.

LOCATE is then called with the sharer's FQN that is in the parameter list. If a X'04' return code is received, processing terminates and a X'04' return code is sent to the caller. If a X'00' return code is received, a check is made to see if the user is trying to build a sharing descriptor in a generation index. If so, processing terminates with a X'18' return code. A check is made to see if the user has already shared the data set. If he has, the caller is sent a X'00' return code. Otherwise a X'08' return code, indicating that the name is not unique, is returned. A X'0C' return code means that a subset of the name is the name of a previously created sharing descriptor and the caller receives a return code of X'08'.

A X'08' return code means that only a subset of the name was found. The catalog entity on which the search terminated could be either an index or a data set descriptor; the SBLOCK retrieved by LOCATE is checked to determine which. If the search terminated on an index, the name is unique and SHAREUP continues to build a sharing descriptor. If the search terminated on a data set descriptor, there can be no additional subordinate levels; thus the name is not unique and SHAREUP returns with a return code of '08'. If the SBLOCK returned is a generation index, SHAREUP returns with a return code of '18'.

If the name is unique, INDEX is called to construct the necessary levels for building a sharing descriptor. INDEX builds all required levels including the level that the sharing descriptor occupies. On return from INDEX, it is necessary to change the last level created by INDEX into the completed sharing descriptor. A nonzero return code causes SYSER to be invoked, since all conditions causing nonzero return codes should have been discovered by the previous LOCATE.

Before the sharing descriptor is built, a check is made to see if INDEX added the new levels to the catalog correctly. If an error is detected, the CATALOG ERROR PROCESSOR is called to claim a SYSER and a comp code 1 ABEND. If the new levels are satisfactory, the sharing descriptor is built and the catalog is updated externally with a call to CZCFG4.

If the final return code is X'00', LOCATE is called again with the sharer's name to see if the sharer is allowed to access the data set. If no access is permitted the caller receives a X'0C' return code.

LOCATE (CZCFL)

LOCATE is a reentrant, privileged routine that resides in public storage. The first entry point (CZCFL1) is called to retrieve SBLOCKs from the user catalog, either by fully qualified name (FQN) or relative address. Conversion of relative numbers (for example, NAME(+1)) to absolute generation numbers (for example, NAME.G0001V00) is performed when required. The second entry point (CZCFL2) is called to locate the catalog level and any lower levels implied for an FQN. Catalog information is placed in one or more TBLOCKs (logical entities containing output catalog data which are located in a GETMAIN area) for each terminal level found in the catalog.

Entry Points:
CZCFL1 -- for LOCATE
CZCFL2 -- for LOCFQN

Input: LOCATE has two separate sets of input parameters. Register 1 contains the address of the following parameter list:

For LOCATE (CZCFI1)

Word 1 Pointer to a fully qualified name or relative address, depending on the option code specified in Word 2

Word 2 Option pointer

- 0004 -- locate on fully qualified name and unlock
- 0008 -- locate on relative address and unlock
- 000C -- locate on fully qualified name and hold interlock
- 0010 -- locate on relative address and hold interlock
- 0014 -- release interlock

Word 3 Pointer to SBLOCK return area (64-byte buffer)

Word 4 Pointer to owner's name and flag return area (45-byte buffer)

For LOCFQN (CZCFL2)

Word 1 Pointer to a fully qualified name

Word 2 Pointer to a fullword area for the return of the number of TBLOCK pages

Word 3 Pointer to a fullword area for the return of the first TBLOCK page virtual memory address

Word 4 Pointer to a 1-byte option code

01 -- cross catalog when necessary

02 -- cross catalog implicitly

03 -- don't cross catalog

Output: Register 1 contains a pointer to the input parameter list.

Modules Called:
GETSBLOCK (CZCFG1) -- To calculate the SBLOCK virtual memory address.
(CZCFG3) -- To provide the initial sizing of the VAM buffer used in processing the user catalog.
(CZCFG4) -- To effect the necessary PUT/PUTX/STOW and CLOSE (TYPE T) sequence.

FIND (CZCOJ) -- To search the POD or entry.

STOW (CZCOK) -- To replace the member descriptor and to close the member.

ERROR PROCESSOR (CZCFE) -- To record a catalog structure error in SYSLOG and issue SYSER and/or ABEND.

DSCB/CAT RECOVERY (CZUFX1) -- To rebuild the catalog in SYSCAT.

GETMAIN (CZCGA2) -- To reserve pages of virtual storage.

READ (CZCPE) -- To read a record from SYSSVCT.

WRITE (CZCPE) -- To write a record in SYSSVCT.

FINDJFCB (CZAEB) -- To find a JFCB for the user catalog.

OPEN (CZCLA) -- To open the USERCAT DCB.

GET -- To bring the user catalog into the buffer.

PUT -- To move the user catalog into SYSCAT.

CLOSE (CZCLB) -- To close the user catalog DCB.

CATFLUSH (CZCFX) -- To remove the catalogs for inactive member from SYSCAT.

INDEX (CZCFI) -- To build the first page of a catalog for a userid.

Exits: LOCATE (CZCFL1)

Normal - register 15 contains X'00'.

Error - register 15 contains one of the following hexadecimal codes:

04 - The userid supplied to the FIND macro instruction is not in the POD of the catalog data set. If the owner's userid is not found, a flag is set in the first byte of the 45-byte buffer, and the owner's fully qualified name is in the remaining 44 bytes.

08 - Not all the qualifiers of the fully qualified name were located. The last qualified SBLOCK located is in the 64-byte retrieval area. If an owner's catalog was entered, a flag is set in the first byte of the 45-byte buffer and the owner's fully qualified name is in the remaining 44 bytes.

0C - The user is attempting to retrieve an entity in an owner's catalog that he is not allowed to share. No SBLOCKs are retrieved in the 64-byte buffer, nor is the owner's fully qualified name inserted in the 45-byte buffer.

LOCFQN (CZCFL2). Same exits as for CZCFL1.

Same as above.

LOCATE calls the ERROR PROCESSOR (CZCFE) to issue a SYSER for any of the following conditions:

1. 80102501 - The FQN in the parameter list contains an error (for example, the userid or qualifier exceeds 8 characters in length).

2. 80102502 - An FQN generation data set name contains an error (for example, if there is a non-numeric character between parentheses).

3. 80102503 - There is an invalid option code in the input parameter list.

4. 80102504 - An unexpected return code is returned from FIND.

5. 80102506 - Another catalog member has to be opened thus forcing LOCATE to implicitly release the current member which has been locked up for an update.

6. 80102507 - The userid is not in the first SBLOCK of a catalog.

7. 80102508 - A catalog cannot be accessed (for example, the member cannot be located by FIND or a STOW cannot be issued successfully).

8. 80102509 - A SYNAD was taken on a READ or WRITE operation on SYSSVCT for a reason other than that the key was not found.

9. 80109902 - This is a general SYSER that CZCFE issues when requested to issue a SYSER but the parameter list to CZCFE does not contain a SYSER. The following conditions in LOCATE cause this SYSER:

   The number of qualifiers exceeds 19.

   The primary catalog buffer has been destroyed after processing an owner's catalog.

   There was an error return from last call to GETSBLOCK (CZCFG1) before exit.

10. 80109901 - This SYSER is issued by CZCFE when LOCATE detects a catalog structure error.

Operation: LOCATE (CZCFL1)

LOCATE is entered with a request to retrieve an SBLOCK either by its fully qualified name (FQN) or its relative address.

After initialization, the option code is examined. If it is not zero or a multiple of four a SYSER and ABEND will be issued informing the user of an invalid option code. If the input option requests a LOCATE on an FQN, a trace table is built. A 24-byte entry is made for each qualifier in the FQN. First, the qualifier is moved into the table after it is validated. A flag is set in the entry to indicate that the level is an explicit one and a second flag is set if the FQN points to a data set in an owner's catalog. A third flag is set for the last qualifier in the FQN. If the FQN is a relative generation, it is converted to binary and the qualifier for the current level contains the binary number followed by a plus or minus sign. A SYSER and ABEND are claimed if any irregularity is detected in the FQN. The trace table is updated when the SBLOCK for the level is found in the catalog. Sharing information and the type and relative address of the SBLOCK are saved in the table. A flag is also set to indicate that the qualifier was found in the catalog.

LOCATE calls CZCFG3 after issuing the FIND against the user's catalog, to initialize the VAM buffer in which the catalog is processed. LOCATE must also determine if the member for which the FIND is to be issued is already open. If it is, LOCATE bypasses issuing the FIND and the call to CZCFG3. If a member is open but does not correspond to the member for which the FIND is to be issued, CZCFG4 is called to issue the necessary PUTX, PUT, STOW and CLOSE (TYPE=T) sequence. Before calling CZCFG4 a check is made to see if the catalog is locked up. If it is, a SYSER and ABEND are claimed and the user receives the message: LOCATE HAD TO IMPLICITLY RELEASE A MEMBER LOCKED UP FOR UPDATE.

If the return code from FIND indicates that the userid in the FQN is not in the SYSCAT POD and if the attempted access is to his own catalog, the address of the user catalog is extracted from SYSSVCT with a READ operation. If a SYNAD is taken on the READ because the key is not found, a return with RC='04' is made. If a SYNAD is taken on the WRITE operation or on the READ operation for any other reason than because the key can not be found the program will issue a SYSER and ABEND and the user receives the message; UNRECOVERABLE ERROR IN SYSSVCT. A STOW 'N' is issued to create a member entry in the POD and this is followed by a FIND to open the member of SYSCAT. If the STOW fails because the POD is full, CATFLUSH is called to delete inactive members from the SYSCAT POD and a branch is taken to reissue the STOW 'N' again. If the CATFLUSH return code is not zero, LOCATE terminates with a '04' return code. If the return code from FIND is other than '00' or '14' (member already open), SYSER 80102508 and an ABEND are issued. The user receives the message: SYSTEM FAULT: UNABLE TO ACCESS CATALOG. If the FIND is successful and the DCB indicates there are pages in the SYSCAT member, a branch is taken to continue normal processing in LOCATE. If there are no pages in the SYSCAT member and SYSSVCT indicates a USERCAT does not exist, INDEX is called to create the first page of a catalog in SYSCAT. The flag UCTSYNC in SYSSVCT is then set to indicate that the SYSCAT member and the USERCAT are not the same by issuing a WRITE or SYSSVCT. Then a branch is taken to continue normal processing.

If there is no SYSCAT member but there is a USERCAT, the USERCAT is moved into SYSCAT. FINDJFCB is called to get the JFCB for the USERCAT. The USERCAT DSCB pointer

is moved from SYSSVCT to the JFCB, TDTVPY is set to indicate that the USERCAT is a privileged data set and the DCB for USERCAT is opened. A GET moves the USERCAT to the buffer, a PUT moves it to SYSCAT and the DCB is then closed. A STOW is executed to close the catalog member. If the return code from STOW is not zero, a SYSER and an ABEND are issued and the user receives the message: SYSTEM FAULT: UNABLE TO ACCESS CATALOG. Before returning to reissue the FIND, SYSCAT is checked to see if it exceeds 800 pages. If it does, CATFLUSH is called to delete inactive SYSCAT members.

After the FIND is complete, CZCFG1 is called to obtain the virtual memory address (VMA) of the first SBLOCK in the catalog. If the return code from CZCFG1 is not zero or if the FQN userid is not in the first catalog SBLOCK, a SYSER is claimed and the ERROR PROCESSOR (CZCFE) is requested to call CZUFX to rebuild the catalog. CZCFE then issues a completion code 1 ABEND after CZUFX returns control.

When the catalog SBLOCK for one of the FQN qualifiers is found, the corresponding entry in the trace table is updated as described above. If this is the SBLOCK for the last qualifier in the FQN and it is not a sharing descriptor, the catalog search is finished. If LOCATE is called by LOCFQN, LOCATE branches back to LOCFQN. Otherwise the caller receives a zero return code and LOCATE exits. The exit procedure consists of moving the last located SBLOCK into the calling routine's 64-byte buffer (given as an entry parameter to LOCATE). This SBLOCK may be an index, a data set descriptor or a sharing descriptor. Sharing information extracted from the current or a higher level is returned in the 64-byte buffer (if a sharing descriptor is not being returned). LOCATE also fills in CDSNPT, CDSCLS and CDSCLC in the catalog common area (CHBCDS). If the input option requests the closing of the catalog member, CZCFG4 is called to close it before returning to the caller.

If the SBLOCK is an index SBLOCK (and there are more FQN qualifiers to be located), a search is made in this index level for the pointer entry containing the next FQN qualifier. If found, CZCFG1 is called to obtain the VMA of the SBLOCK and a branch is made to update the trace table as described previously. If the next FQN qualifier is not found in the index level, the caller receives an X'08' return code. If LOCATE is called by LOCFQN the module branches to it. If not the module exits as described previously.

If the SBLOCK is a sharing descriptor and processing is already in an owner's catalog, a X'0C' return code is sent to the caller or LOCFQN. If LOCATE was called by DELCAT, the catalog search is complete and this sharing descriptor is returned to DELCAT. Otherwise LOCATE moves the owner's FQN from the sharing descriptor to the calling routine's 45-byte buffer. Any remaining qualifiers in the original FQN are then concatenated with the owner's FQN in the buffer. If adding these qualifiers causes the FQN to exceed 44 characters a SYSER and ABEND are claimed. The sharing descriptor is then moved into the 64-byte buffer in case the owner userid is not found. If the owner and sharer userids are not the same the current catalog member is closed before branching back to build a new trace table for the owner's FQN. Before closing the catalog member a check is made to see if the caller requested LOCATE to keep the catalog locked until it is updated (CDSLOC flag was set). If so, a SYSER and ABEND are claimed and the user receives the message: LOCATE HAD TO IMPLICITLY RELEASE A MEMBER LOCKED UP FOR UPDATE.

If the catalog level pointed to by the owner's FQN is successfully retrieved, and the user is allowed to share, a flag (X'04') is set in the first byte of the 45-byte buffer to indicate that an owner's catalog was entered and LOCATE will exit with a zero return code. If the user is not allowed to share the owner's data set, the buffer flag is set to zero and LOCATE exits with a X'0C' return code.

When LOCATE is given a fully qualified name containing a relative generation number, the relative generation number is converted to an absolute generation name. The last qualifier before the relative number must point to a generation index, or LOCATE terminates with an error condition. When the absolute name associated with a zero or negative relative number is located, the relative number is overlayed in the fully qualified name by the absolute name. If a positive number is given, a new absolute name is generated by adding the last generation number to the relative number. The result is put in the fully qualified name, as with zero or negative numbers. LOCATE then proceeds as usual, retrieving the data set descriptor for zero or negative numbers, and returning an error code for positive numbers. If the generation index is in an owner's catalog, the relative number is replaced in the 45-byte buffer, not in the original fully qualified name.

Sharing information is extracted from the catalog by searching all levels of the catalog until an access is found. The search begins at the lowest level (the last level found) and can continue up to the highest (userid). The first access that is found is returned. If a level is shared

selectively, the sharing list is searched for the userid. If a level is shared universally, that access is returned.

A request by relative address may be made only after a LOCATE has been executed for a fully qualified name and it is necessary to retrieve subsequent chained SBLOCKs. However, if any intermediate request causes LOCATE to enter another user's catalog, a request by relative address is not valid for any relative address within the first user's catalog. When this option is used, no fields in the SBLOCK are changed.

LOCATE checks for catalog structure errors when searching through a catalog. If an error is detected, the CATALOG ERROR PROCESSOR (CZCFE) is called with an error code to describe the type of error and with other data such as the location of the error in the catalog. If LOCATE is called by LOCFQN a flag may be set requesting CZCFE to return after claiming a SYSER and processing continues in LOCATE. Otherwise CZCFE issues an ABEND.

Note: LOCATE's PSECT (CZCFLX) contains a data control block (DCB) for SYSCAT.

LOCFQN CZCFL2

LOCFQN is entered to find the catalog level and any lower levels implied for an FQN. This information will be returned to the caller in 96-byte blocks called TBLOCKs. The TBLOCKs are all chained together on storage obtained by a GETMAIN operation. The caller must free this storage when the TBLOCKs are no longer required.

LOCFQN calls LOCATE (CZCFL1) with the FQN in the input parameter list. Upon returning from LOCATE the return code is saved and a switch is set to activate error recovery if an error is encountered during a lower level implicit search of the catalog. If an error is detected a call is made to the ERROR PROCESSOR (CZCFE) to issue a SYSER and ABEND. If the flag is set the ERROR PROCESSOR may be requested for certain recoverable situations to return to the caller after issuing the SYSER.

If a zero return code is obtained from the LOCATE call the virtual memory address (VMA) of the catalog SBLOCK associated with the last qualifier in the FQN is obtained by calling GETSBLOCK (CZCFG1). If a non-zero return code is obtained for GETSBLOCK, the ERROR PROCESSOR is called to issue a SYSER and ABEND and the user will get the message: SOFTWARE ERROR-ERROR RTN ON LAST GETSBLOCK BEFORE EXIT.

The SBLOCK is then analyzed to determine its type. If it is a data set descriptor SBLOCK and a entry was not made into an owner's catalog, a TBD type TBLOCK will be built. TBS TBLOCKs containing permissive information for all qualifier levels of the FQN will also be built and chained to the TBD TBLOCK. If entry was made into an owner's catalog a TBD TBLOCK is built only if the sharer has access to the data set. After building the TBLOCK for this level in the FQN, a check is made to see if this qualifier level is in the FQN in the parameter list to LOCFQN. If so, it is the end of the processing in the current catalog and the catalog is closed. This is the same proceedure followed when a return code of X'08' or X'0C' is received from the initial call to LOCATE. If the closed catalog is not an owner's catalog, processing is complete and a return is made to the caller.

If processing is in an owner's catalog, a check is made to see if the sharer's catalog was previously closed. This is the same procedure to be followed for a return code of X'04' from the initial LOCATE call. If it was closed the buffer is checked to see if the first SBLOCK in the catalog contains a userid. If it does not, a SYSER is claimed and the ERROR PROCESSOR is asked to return instead of issuing an ABEND. If there had been an explicit search into an owner's catalog but no TBLOCKs were built because of a lack of sharing information, an initial return code of X'00' from LOCATE is changed to X'0C' before exiting from the module. If an explicit search into the owner's catalog yields no output, a TBD TBLOCK is built containing the sharer's FQN. A TBO TBLOCK is also built containing the owner's FQN and a flag, which is set to explain why there is no output.

The module then checks to see if the current qualifier level is explicit (that is, the qualifier is in the input FQN). This procedure is also followed after output of a TBD TBLOCK. If the level is explicit, processing in this catalog is complete and the member is closed as previously described. If not, a search is made for the next lowest qualifier. If none is found, the entry in the trace table (see LOCATE for explanation of trace table) for this qualifier is erased and the module will back up to the next higher qualifier to see if it is explicit. When a qualifier is found a check is made to see if the number of qualifier levels exceeds 19. If so, a SYSER is claimed, all the lower qualifier levels are ignored and a TBD TBLOCK is built as described above. If the number of qualifier levels is fewer than 19, a new trace table entry is built for this qualifier level and the module branches to analyze the type of SBLOCK in the catalog for

this level as described after the initial call to LOCATE.

If the SBLOCK that is returned after the initial call to LOCATE is a sharing descriptor, a check is made to see if an owner's catalog has been entered. If so, LOCFQN backs up to examine the next higher qualifier as described above. If an owner's catalog has not been entered and LOCFQN is requested not to enter a catalog, a TBD and TBO TBLOCK are built and LOCFQN backs up again to examine the next higher qualifier. If LOCFQN is requested to implicitly enter a catalog, a flag is set indicating an owner's catalog has been entered and the sharer's catalog is closed. Since it will not be reopened again, each page in the buffer is marked as changed so that page-ins can be done from the drum or auxiliary storage instead of external storage which could be changed since the catalog is closed. If sharer and owner userids are the same the catalog is not closed. LOCATE is now entered for the second time using the FQN in the sharing descriptor. Finally, if none of the above options is requested, LOCFQN enters an owner's catalog only if the qualifier level which points to the sharing descriptor is in the input FQN (that is, is an explicit qualifier level). Otherwise a TBD and TBO TBLOCK are built before branching back to look at the next higher qualifier.

If the SBLOCK that is returned after the initial LOCATE call is an index, LOCFQN looks for the first member in this index level. If none can be found, a TBD TBLOCK is built as described previously. If a member is found, LOCFQN branches to check if the number of qualifier levels exceeds 19.

A TBD TBLOCK contains an FQN (TBDDSI), the DSCB pointer if it is a VAM data set (TBDDSC), the owner's or sharer's access to the data set (TBDACC), the owner's userid if the data set is shared (TBDOWN) and data set organization (TBDORG). The TBLOCK has room for two volume fields (TBDDVF, TBDDVL) for a SAM data set. If the data set exists on more than two volumes, this information is stored in TBC TBLOCKs which are chained to the TBD TBLOCK. A TBDFLL flag is also set if the data set has BULKIO pending for it or if it is a temporary data set.

If a TBD TBLOCK is built for a sharing descriptor, either because of the input option, or because of an unsuccessful search into the owner catalog, a TBO TBLOCK is also built and chained to the TBD TBLOCK. The TBO TBLOCK contains the owner data set name issued by the user at SHARE time and flags, if applicable, indicating the reason for the unsuccessful search (for example, userid not found, data set non-existent or non-sharable).

Sharing information is stored in TBS TBLOCKs which are chained to the TBD TBLOCK. Each entry in the TBLOCK (ten bytes) consists of a userid, the access, and the FQN level associated with the access. If a level is shared universally, the userid is set to *ALL.

CATALOG ERROR PROCESSOR (CZCFE)

The CATALOG ERROR PROCESSOR is a reentrable, nonrecursive, privileged subroutine residing in virtual storage. Its function is to execute catalog SYSERs and to take action when a catalog structure error or a user format error is discovered by one of the catalog service routine modules. If the module is called to execute a catalog SYSER (error codes F0-FF) the SYSER and ABEND in the parameter list are checked and then issued from this module. When a catalog structure error (error codes 00-9F) is discovered by one of the catalog service modules, the ERROR PROCESSOR is called with the type and location of the SYSER in the catalog. This information is recorded in the system log and a SYSER and an ABEND are then issued. If a user format error (error codes E0-EF) is detected, the error processor is called to issue an ABEND to the user.

Entry Point: CZCFE1

Input: The parameter list to the module is in the PSECT CBBCEP. It contains the following information:

Word 1/Word 2 Module name of caller

Word 3: Contains the following:

- byte 1 - hexadecimal error code
- byte 2 - exit option code
- byte 3 - flag byte (not used)
- byte 4 - not used

Word 4 Address of FQN (44-byte area).

Word 5 Address of qualifier level at which error occurs (this address should be in the range of the FQN).

Word 6 Address of minor SYSER in caller's PSECT.

Word 7 Address of ABEND message in caller's PSECT (the first byte gives the length of the text that follows).

Word 8 Primary address of the error.

Word 9 Secondary address associated with the error.

Word 10 Contains the following:

bytes 1,2 - actual count of entries
byte 3 - not used
byte 4 - not used

Output: None

Modules Called: SYSER (CEAIS) -- system error routine: invoked when the error processor is called with errors codes 00 to 9F and E0-EF.

Exits: Return, to the caller, if specified by the option code (CEPOPT). Otherwise, issues an ABEND.

SYSERs:

80109901 This SYSER is issued when CZCFE is called when one of the catalog service routines discovers a catalog structure error (error code 00-9F)

80109902 This SYSER is issued when CZCFE is called to issue a SYSER but the parameter list does not contain one (error code F0-FF)

80109903 This SYSER is issued if the parameter list contains a SYSER that is not type 1 (error code F0-FF).

Operation: The CATALOG ERROR PROCESSOR is called for three types of errors detected by the Catalog Service Routines:

1. Type 1 (error codes 10-9F) a catalog structure error.

2. Type 2 (error codes E0-EF) a user input format error with no appropriate return code available for returning to the caller.

3. Type 3 (error codes F0-FF) other system error conditions for which the caller supplied a SYSER code and a comp code 1 ABEND message.

If the parameter list contains a fully qualified name (FQN), it is added to all ABEND and WTL messages. If the name exceeds 44 characters only the first 44 are added. If the parameter list contains an address of a qualifier that is in the FQN range, the qualifier is also added to the messages.

When called for a type 1 error, the CATALOG ERROR PROCESSOR checks to see if the parameter list contains a SYSER. If it does not, SYSER 80109902 is executed. If the SYSER is not a type 1 SYSER, the module issues a 80109903 SYSER. Otherwise the SYSER in the parameter list is issued. If the parameter list contains an ABEND message, the FQN and the calling routine are added to it before writing the message to SYSLOG. If the parameter list does not contain an ABEND, a standard message is written in SYSLOG instead. Before issuing the ABEND (either the one in the parameter list or the standard ABEND message), the option code is checked. If it specifies a return, the CATALOG ERROR PROCESSOR returns to the caller. If it specifies that the catalog is to be rebuilt, CZUFX is called to do this before return is made to the caller. Otherwise the ABEND message is issued.

When called for a type 2 error the CATALOG ERROR PROCESSOR checks the parameter list to see if it contains an ABEND message. If it does, the message is concatenated with the error code, the module name of CZCFE's calls an FQN. Then a comp code 1 ABEND is issued. If the parameter list does not contain an ABEND, a standard message is issued.

If called for a type 3 error, the CATALOG ERROR PROCESSOR writes a message to SYSLOG describing the type and location of the error in the user's catalog. The FQN and the module which discovers the error are also included in the message. The option code is again checked. If a return is requested, the module returns to the caller. If it requests that the catalog be rebuilt, CZCFX is called to do this before returning to the caller. Otherwise, a comp code 1 ABEND is issued and the user receives a standard message along with the error code, the module which detected the error and the FQN.

Note: CZCFE8 is the location in CZCFE where all SYSERs are executed. CZCFER is the location of the area in CZCFE's PSECT which contains the write-to-log message.

INDEX (CZCFI)

INDEX is a reenterable, nonrecursive, privileged subroutine residing in virtual storage, that constructs chained index levels in the catalog and creates new members within the catalog data set. The fully qualified name input parameter is inspected to determine if a new member is being created or a new index level is to be added to the user. If the FQN consists of just one component, INDEX will build the userid SBLOCK in the catalog and exit. When index levels are to be chained for the user, the lowest level found is searched for an empty pointer and a pointer is constructed to the first SBLOCK of the level being created. (See Chart AI.)

Entry Point: CZCFI1

Input: Register 1 contains a pointer to the parameter list:

Word 1 Pointer to fully qualified name

Word 2 Pointer to special parameters

bits 0-7 - generation code 40; create generation index

bits 8-15 - maximum generation number

bits 24-31 - generation flags

Output: Register 1 points to the input parameter list.

Modules Called:
GETSBLOCK (CZCFG1) -- To locate SBLOCK and calculate virtual storage address.
(CZCFG4) -- To update the catalog.

SEARCHSBLOCK (CZCFH) -- To acquire and chain an empty SBLOCK.

PUT (locate mode) (CZCOS) -- To locate a buffer to be the next record put in the data set.

LOCATE (CZCFL) -- To locate an index level.

CATALOG ERROR PROCESSOR (CZCFE) -- To claim a SYSER and a completion code 1 ABEND when a catalog structure error is detected, or when an end-of-data-set condition is detected by CZCFG1. CZCFE is also called whenever a SYSER is claimed because the userid in FQN is not 8 characters in length.

Exits:
Normal - register 15 contains 00 - a new member was successfully added, or all qualifiers (index levels) not previously cataloged were added.

Error - if INDEX was unsuccessful, register 15 contains one of the following codes:

04 - userid not found in POD.

08 - all components of the fully qualified names were already in the catalogs, or the generation name had a format error.

0C - user is attempting to update a user catalog for which he is not authorized.

Operation: INDEX inspects the fully qualified name supplied as an input parameter to determine whether the name has one component or more than one. If there is only one component (the userid), the PUT macro instruction is issued to acquire a buffer in which to construct the userid SBLOCK. INDEX returns with a X'00' return code after building the userid SBLOCK. The STOW macro instruction is issued in LOCATE.

If the fully qualified name is composed of more than one component, INDEX will enter LOCATE to determine if the same name already exists. If LOCATE indicates that all the components of the fully qualified name were found (code 0), INDEX will return a code of '08' to the calling routine. A return code of '04' or '0C' is passed on as a return code from INDEX, and no further processing takes place.

Upon locating some, but not all of the levels of the fully qualified name, INDEX determines, in the case of a sharer, if the sharer is privileged to update the catalog. If not, an error code of '0C' is returned. If the lowest level found is a generation index, the next component of the fully qualified name must have the absolute generation name format. Otherwise, an error code of '08' is returned.

The lowest level found is then searched for an empty pointer. If none exists, the SEARCHSBLOCK routine is entered to acquire and chain an empty SBLOCK in which a pointer is to be placed. This is a pointer to the first SBLOCK of the level being added. This process is continued until the entire fully qualified name has been cataloged.

If the parameter list indicates that a generation index is being created, the generation options are put in the SBLOCK of the lowest level created.

When the level is being added to an existing generation index, the pointer to the new level is sorted by generation number into the list of pointers belonging to that generation index.

Validity checks are performed on data in the catalog when INDEX is searching the catalog. If an error is detected, the CATALOG ERROR PROCESSOR is called with parameters describing the type and location of the error in the catalog, to claim a SYSER and a completion code 1 ABEND.

A call to CZCFG4 is made to update the catalog on external storage.

GETSBLOCK (CZCFG)

GETSBLOCK is a reenterable, nonrecursive, privileged subroutine residing in virtual storage. When entered at its primary entry point (CZCFG1), it receives a pointer containing the relative address of an SBLOCK and calculates its virtual storage address for the user. The user submits the pointer to the desired SBLOCK in the format 0Pbb, where P is the page number relative to the member and bb is the relative byte within the page. If the requested SBLOCK is in the page buffer, the virtual storage address is calculated and returned to the user. If the P value exceeds the size of the user's catalog, CZCFG2 is called which returns to the user with a return code of X'04'.

Entry point CZCFG3 is invoked by LOCATE to provide initial sizing of the VAM buffer used in processing the user catalog.

Entry point CZCFG4 is used to perform the PUT or PUTX functions as indicated by the bit settings in the CHACDS table. (See Chart AF.)

Entry Point:
CZCFG1 - Primary entry point. Used to convert a 'Pbb' address into a 32-bit Virtual Memory address.
CZCFG2 - EODAD exit.
CZCFG3 - Entry point from LOCATE (CZCFL) to provide initial sizing of the VAM buffer used in processing the user catalog.
CZCFG4 - Used to effect PUTX/PUT/STOW/CLOSE sequence as indicated by the bit settings in the CHACDS table (CDSFLG).

Input: Register 1 contains a pointer to the relative address of the requested SBLOCK.

Output: Register 1 contains a pointer to the virtual storage address of the requested SBLOCK.

Modules called:
SETL (CZCOT) -- To locate the specified page of a catalog member.

GET (CZCOR) -- To move the specified page to the catalog buffer.

STOW (CZCOK) -- To unlock a member in the POD.

CLOSE (CZCOQ) -- To disconnect a data set from the system.

PUT (CZCPA) -- To add a record to an output data set.

PUTX (CZCOU) -- To exchange a record in an output data set.

READ (CZCPE) -- To read a record of SYSSVCT.

WRITE (CZCPE) -- To write a record of SYSSVCT.

CATALOG ERROR PROCESSOR (CZCFE) -- To issue any SYSERs and ABENDs claimed by GETSBLOCK.

Exits: If the return code from STOW is not zero a SYSER is issued followed by an ABEND. If end-of-data set occurs, control is returned to the user with 04 in register 15; otherwise, control is returned with 00 in register 15.

Operation: CZCFG1 - This is the primary entry point used to convert a 'Pbb' address into a 32-bit Virtual Memory address. This is done by multiplying (P-1) by 4096, and adding the product to the beginning of the user's catalog buffer. Adding the bb value yields the required Virtual Memory address.

No physical GET operations are performed. If the P value exceeds the size of the user's catalog, CZCFG2 is entered.

CZCFG2 - This is the EODAD exit after a GET or PUT is issued for a catalog member. A SYSER and ABEND will be claimed if the EODAD occurs.

CZCFG3 - This entry point is invoked by LOCATE (CZCFL) after the call to FIND (CZCOJ), to provide initial sizing of the VAM buffer used in processing the user catalog. The buffer must be at least one page greater than the current size of the member (DCBDMS). If the buffer is too small, a FREEMAIN and GETMAIN sequence is performed to release the current buffer and obtain the required space for the new buffer, storing these parameters in the DCB Header. DCBEXL, DCBBCN, DHDDXP, and DHDCPR are cleared to ensure that paging of the member will be effected on a subsequent GET operation.

The DHDMRL field is set to one page greater than the current size of the member. This field is set to ensure proper execution of CZCFG4 and the edit performed by SEARCHSBLOCK (CZCFH).

A SETL is issued to the beginning of the member, followed by a GET (locate mode). The address returned by GET is stored in BUFFAD and used by CZCFG1 to calculate the Virtual Memory address requested.

CZCFG4 - This entry point is used to effect the necessary PUTX/PUT as indicated by the bit settings in the CHACDS table (CDSFLG). The CDSFLG bits are then reset, and a STOW

(type R) is issued against the DCB pointed to by CZCFL2. The CZCFL2 field is reset to point to the primary DCB (CDSCOM).

If a PUTX was issued, a check is made to see if the flag byte in the SYSSVCT record indicates that the USERCAT and the SYSCAT (SCRATCHCAT) members are the same. (They should not be the same since the PUTX changed SYSCAT.) If the flag indicates the catalogs are the same, the record is reset and written back into SYSSVCT. If a SYNAD is taken on a READ or on a WRITE, the CATALOG ERROR PROCESSOR is called to issue a SYSER and ABEND and the user receives the message: UNRECOVERABLE ERROR IN SYSSVCT.

If a PUT is issued against a member due to an added page, a CLOSE (TYPE=T) is issued instead of the STOW.

SEARCHSBLOCK (CZCFH)

SEARCHSBLOCK is a reenterable, nonrecursive, privileged subroutine residing in virtual storage. It acquires and chains an empty SBLOCK as either:

- An extended SBLOCK of a cataloged entity.
- The first SBLOCK of a cataloged entity.

The count of SBLOCKs in each page is checked until an available SBLOCK is found. The relative address of the SBLOCK within the page is then located by searching for a control byte of zeros. The new SBLOCK is retrieved by GETSBLOCK and its virtual storage address is returned to the user. The new SBLOCK is linked to the parent SBLOCK before returning control to the user. (See Chart AG.)

Entry Point: CZCFH1

Input: Register 1 contains a pointer to the address of the SBLOCK to which the new SBLOCK is to be chained.

Output: Same as input

Modules Called:
GETSBLOCK -- locate SBLOCK and calculate virtual storage address.

CATALOG ERROR PROCESSOR (CZCFE) -- To claim any SYSERs and comp code 1 ABENDs issued by SEARCHSBLOCK.

Exits:
Normal - return without return code

Error - to SYSER without return code

Operation: SEARCHSBLOCK scans the existing pages of the user's catalog. By testing the count of SBLOCKs in the page (64 is the maximum), a page containing an available SBLOCK can be located. When an end-of-data set condition is encountered while scanning through the pages, SEARCHSBLOCK appends a new page to the user's catalog through use of a simulated PUT (locate mode). SEARCHSBLOCK sets the PUT indicator in the CHACDS table, calculates the VMA of the new page by adding the current value of LRECL to that of DCBBCN, and then increases the value of LRECL by 4096. The new page is zeroed out and thereby initialized.

Once the page is retrieved, SEARCHSBLOCK scans the allocated byte field of each SBLOCK in the page. If this field is zero, the SBLOCK is free; if this field is not zero, the next SBLOCK is examined. This procedure is repeated until an empty SBLOCK is found. SEARCHSBLOCK inserts a backward pointer in the new SBLOCK, and retrieves the parent SBLOCK to which it is chained. The forward chain in this SBLOCK can be one of the following:

- Extended SBLOCK pointer.
- Pointer to a list of sharers.
- Pointer field in a member of an index level.

SEARCHSBLOCK determines the proper field and inserts the forward chain.

The new SBLOCK is retrieved via GETSBLOCK and control is returned to the calling program.

USERCAT SCAN (CZUFY)

USERCAT SCAN is a reenterable, nonrecursive, priviliged subroutine that resides in virtual storage. It is called at entry point CZUFY1 when OPEN VAM or DSCB/CAT RECOVERY discovers an error while reading the DSCB for the SYSSVCT data set. The first data page of the data set is examined to see if it contains records for userids TSS*****, SYSOPER0, and SYSMANGR, and if the locators at the bottom of the VISAM page for the records are correct. If all these conditions are true, the checksum is computed for the SYSSVCT DSCB, and the DSCB page is written out before leaving the module. Otherwise, the first page of SYSSVCT is constructed with records for the three userids and the page is written onto a direct access device. The DSCB for SYSSVCT is initialized, the checksum is computed, and this page is written out. The operator is then prompted with a message telling him to call CZUFY2 after startup is complete.

The data set SYSSVCT is rebuilt at entry point CZUFY1. First, the DSCB pointer for

each record in SYSSVCT is filled with zeros. The public volumes are then scanned for usercat DSCBs. When found, the DSCB pointer is computed and entered into SYSSVCT for the userid. (See Chart AJ.)

Entry Points:
CZUFY1 - To compute checksum for SYSSVCT DSCB and to construct the first page of SYSSVCT data set if necessary.
CZUFY2 - To initialize SYSSVCT and then rebuild it by moving in the DSCB pointer for all of the usercat data sets.

Input: None

Output: None

Modules Called:
SETL (CZCOT) - To position to beginning of SYSUSE data set.

GET (CZCOR) - To retrieve a record in SYSUSE.

OPEN (CZCLAO) - To open SYSUSE and SYSSVCT data sets.

READ (CZCPE) - To read a record in SYSSVCT.

WRITE (CZCPE) - To write a record in SYSSVCT.

PRMPT (CZATJ) - To issue a message to user and operator.

CLOSE (CZCLB) - To close SYSSVCT and SYSUSE data sets.

READWRIT (CZCEM) - To read and write a DSCB page.

ESALOCK (CZCEJ) - To lock and unlock the SDAT PAT lock.

FINDJFCB (CZAEB) - To find the JFCB for SYSSVCT.

DDEF (CZAEA) - To build a JFCB for SYSSVCT.

FINDEXPG (CZCEL) - To get a page of storage on a direct access device.

GETMAIN (CZCGA) - To get a page of virtual storage.

FREEMAIN (CZCGA) - To free a page of virtual storage.

ABEND (CZCAP) - To abort from a module.

Exits: The module does not set a return code on normal exit.

ABEND: If DDEF could not create a JFCB for SYSSVCT.

If FINDEXPG could not get a free page on a direct access device.

If READWRIT could not write out a DSCB page containing the corrected DSCB for SYSSVCT.

If READWRIT could not read a DSCB page containing the DSCB for SYSSVCT.

Operation: The entry point CZUFY1 is called when OPENVAM or DSCB/CATALOG RECOVERY encounters an error in the SYSSVCT DSCB. FINDJFCB is called to retrieve the JFCB. If one is not found, DDEF is called to create one. If a JFCB cannot be created, an ABEND is issued and the user receives the following message:

SYSTEM FAULT: 001, UNABLE TO ACCESS USERCAT

The DSCB pointer is obtained from the JFCB and READWRIT is then called to read the DSCB page into virtual storage. If no error is detected in reading the DSCB page, the module returns to the caller. If an error other than checksum is detected, an ABEND is issued, and the user receives the following message:

SYSTEM FAULT: 002, UNABLE TO ACCESS USERCAT

If a checksum error is detected by READWRIT, the first page of the SYSSVCT data set is read into main storage at CZCOW1 by issuing a SETXP macro instruction. The page is examined to see if it contains records for userids TSS*****, SYSOPER0, and SYSMANGR, and to see if the locators at the bottom of the VISAM page for these userids are correct. If these conditions are true, the checksum for the SYSSVCT DSCB is computed and the DSCB page is written out by calling READWRIT before leaving the module. (ESALOCK has to be called to lock the SDAT PAT page before writing the DSCB page. When the write is completed, ESALOCK must be called again to unlock the SDAT PAT lock.) If the write operation is unsuccessful, an ABEND is issued, and the user receives the following message:

SYSTEM FAULT: UNABLE TO WRITE DSCB OR PAT PAGE

If the first page of the SYSSVCT data set is not correct, the data set is rebuilt with records for userids TSS*****, SYSOPER0, and SYSMANGR on a page of virtual storage obtained by a GETMAIN macro call. The DSCB pointer for the USERCAT of each userid is computed and put into SYSSVCT. The page is then written out to a page on a direct access device which was obtained by calling FINDEXPG. If FINDEXPG can not obtain a page of storage an ABEND is issued

and the user receives the following message:

SYSTEM FAULT: EXTERNAL STORAGE EXHAUSTED

The DSCB for SYSSVCT is then initialized, and the page containing the DSCB is written out as described above. The operator is prompted with a message telling him to call entry point CZUFY2 after startup is complete. FREEMAIN is then called to free the page of storage obtained by calling GETMAIN. The corresponding byte in the PAT page table is found and set to indicate that the page is in use. The PAT page is then written out by READWRIT.

Entry point CZUFY2 is called to rebuild the SYSSVCT data sets. First, the SYSUSE data set is opened if it is not already open, and a SETL macro instruction is issued to position the beginning of the data set. Then, a record is obtained by a GET. A userid is obtained from the SYSUSE record, and it is used as the key to read a record in SYSSVCT after it has been opened. If the read is unsuccessful, the DSCB pointer in the SYSSVCT record will be filled with zeros, a WRITE UPDATE will be issued on SYSSVCT, and the program will get the next record in SYSUSE. If the READ or WRITE UPDATE results in a SYNAD for a reason other than because the userid was not found, the user is prompted with the following message:

UNABLE TO ACCESS USERCAT FOR THE USERID

and the program looks at the next record in SYSUSE. If a SYNAD is taken because the userid was not found in SYSSVCT, a WRITE NEW will be issued on SYSSVCT, and the program will get the next record in SYSUSE. If a SYNAD is taken on a WRITE NEW, the user is prompted with the following message:

USERID CANNOT BE ADDED TO USERCAT TABLE

and again the program branches back to get the next SYSUSE record.

When all the records in SYSUSE have been read, SYSUSE is closed if it was opened in this module. SYSSVCT is now initialized so that all of the records have a zero DSCB pointer. All of the public volumes are then searched for USERCATs by examining the PAT page on each public volume for a DSCB page. When one is found, READWRIT is called to read the page into main storage at CZCOZ1. Each format E DSCB is examined for an FQN of userid.USERCAT. If found, the DSCB pointer is computed and moved into SYSSVCT with a WRITE UPDATE using the userid in the DSCB as the key. If the WRITE results in a SYNAD, the user is prompted with the following message:

UNABLE TO ACCESS USERCAT FOR THE USERID

When all of the DSCBs on all of the volumes have been checked, the module returns.

CATFLUSH (CZCFX)

CATFLUSH is a public, reenterable, privileged subroutine that copies members of the scratch catalog (SYSCAT) into individual user catalogs at task termination, deletes closed members from the scratch catalog when the virtual storage POD becomes full, and erases the scratch catalog at shutdown. (See Chart AL.)

Entry Points:
CZCFX1 - To copy specific members into the user catalog.
CZCFX2 - To copy inactive members into the user catalog and erase them from the scratch catalog (SYSER if erase is impossible).
CZCFX3 - To copy all members into the user catalog and erase the scratch catalog.
CZCFX4 - To copy SYSOPER0 member into the user catalog.
CZCFX5 - SYSSVCT DCB
CZCFX6 - USERCAT DCB
CZCFX7 - SYNAD entry
CZCFX8 - To copy inactive members into the user catalog and erase them from the scratch catalog (no SYSER if erase is impossible).

Input: Register 1 points to a userid. (The parameter list is for entry point CZCFX1 only.)

Output: None

Modules Called:
READ/WRITE (CZCPE) -- To read or write a record into SYSSVCT.

FINDJFCB (CZAEB) -- To find the SYSCAT and USERCAT JFCBs.

DDEF (CZAEA) -- To create a USERCAT JFCB if none is found.

ADDDSCB (CZCEK) -- To allocate a DSCB for a new USERCAT.

OPEN (CZCLA) -- To open a USERCAT DCB.

SETL (CZCOT) -- To position the data set (old USERCAT).

PUT (CZCOS) -- To copy the user catalog from the scratch catalog.

CLOSE (CZCLB) -- To close the USERCAT DCB.

STOW (CZCOK) -- To unlock a scratch catalog member (type R), and to erase a scratch catalog member (type D).

FIND (CZCOJ) -- To lock a member of the scratch catalog.

GETSBLOCK (CZCFG) -- To set up the scratch catalog buffer (entry point CZCFX3).

RELEXPG (CZCEN) -- To give back pages of the scratch catalog at shutdown.

READWRIT (CZCEM) -- To read or write the scratch catalog DSCB.

DSCBREC (CZCEF) -- To handle a DSCB error.

PATLOCK (CZCEJ) -- To lock or unlock a PAT when writing a DSCB.

SYSER (CEAIS) -- To indicate an error condition.

ABEND (CZACP) -- To terminate processing for an error condition.

Exits:
Normal - register 15 will contain a zero return code.

Error - SYSER or ABEND

Operation: The input parameter list (consisting of a pointer in register 1 to an eight-character userid) is required for entry point CZCFX1. All other entry points work on the entire scratch catalog, except CZCFX4 which works only on SYSOPER0. Before a member of SCRATCHCAT is copied to USERCAT, the entry for this user in SYSSVCT is checked for UCTFLG=X'00' or '01'. If this sync byte is '01', no changes have been made to SCRATCHCAT and no update is necessary. If the sync byte is '00', USERCAT is updated from SCRATCHCAT and UCTFLG is set to '01'. (In this description "sync" means the scratch catalog and the user catalog for a member are the same.)

CZCFX1 (entry point 1) is called by LOGOFF or ABEND to update a user catalog. This will ensure that a user's catalog contains any changes made to the scratch catalog. The member is not deleted from the scratch catalog at this time.

CZCFX2 (entry point 2) is called by LOCATE if it is necessary to add a member to the scratch catalog and there is no room in the virtual storage POD to insert the member. If all members are active, a minor SYSER is declared and the task is terminated. Before any member is deleted (by the STOW D macro instruction), the UCTFLG byte is checked for the RCR Ration Flag (UCTFLG=X'02') and if it is set, the member is skipped and not deleted.

CZCFX3 (entry point 3) is called by SHUTDOWN to update all members not already updated and then erase the scratch catalog. When entered at CZCFX3, CATFLUSH checks the RCR Ration Flag for each member and, if it is set for a member, bypasses that member and does not erase the scratch catalog. All successfully updated members are deleted. All members in SCRATCHCAT updated with no RCR ration flag set are also deleted.

To erase the scratch catalog, RELEXPG (CZCEN) is called to release all pages from the scratch catalog except the format E DSCB slot. Then, the DSCB is read into CZCOZ by a call to READWRIT (CZCEM). The following fields in the format E slot are filled with zeros:

DSELBP	bytes in last page
DSENDP	data pages
DSEDOP	directory pages
DSENOP	overflow pages
DSETNP	total pages
DSEENT	(38 full words)
DSECHN	chain pointer

The checksum for the DSCB is then recalculated and written to external storage by READWRIT.

CZCFX4 (entry point 4) is called by MOHR (STARTUP) to force a refresh of SYSOPER0 to ensure that this particular user catalog is updated.

CZCFX8 (entry point 5) is called by LOCATE to reduce the size of the scratch catalog when it reaches 800 data pages. If all members are currently active, a return is made to the caller.

CATFLUSH uses the following method to copy from the scratch catalog to the user catalog. If no user catalog exists (no address in UCTDSCB in SYSSVCT), FIND JFCB is called to locate the JFCB for this user catalog. If no JFCB exists, DDEF is called to create one. ADDDSCB is then called to assign a format 'E' DSCB for this user catalog, the pointer to which is placed in IDTDSC before the call to OPEN. Prior to opening the USERCAT, CATFLUSH ensures that the member to be copied from the scratch catalog is write-interlocked and if it is not, a write interlock is imposed before the user catalog is opened. Read interlocks are cleared before setting a write interlock.

Also before opening the user catalog, CATFLUSH sets the RCR Ration Flag in SYSSVCT for this entry. CATFLUSH then opens the user catalog for output. If the OPEN is successful the DSCB pointer is placed in UCTDSCB and a WRITE is issued. CATFLUSH reads pages from the scratch catalog (by GET macro instructions), writes pages to the user catalog (by PUT macro instructions), and then closes the user catalog.

When a user catalog exists, CATFLUSH moves the DSCB pointer for the user catalog from UCTDSCB to TDTDSC. Before opening the user catalog, CATFLUSH SETS THE RCR Ration Flag in SYSSVCT for this entry. A SETL 'B' is done after the user catalog is opened. The PUT macro instruction that follows will truncate any pages no longer needed, or add pages if necessary. On the successful completion of the PUTs to the user catalog, the UCTFLG is set to X'01' to indicate that the update is complete and the RCR Ration Flag is cleared.

DSCB/CAT RECOVERY (CZUFX)

DSCB/CAT RECOVERY is a public, reenterable, privileged subroutine that provides dynamic error recovery for either the scratch catalog or the user catalogs. It updates a user catalog if the current member in the scratch catalog cannot be used, or rebuilds a user catalog if no member exists in the scratch catalog and the user catalog is unusable. If the user catalog is rebuilt, the user must reenter all sharing information because it is lost in rebuilding from public DSCBs. (See Chart AK.)

Entry Point: CZUFX1

Input: Register 1 contains a pointer to a two-word parameter list that is organized as follows:

Word 1 address of an eight-character userid

Word 2 address of a one-byte flag field

The flag will be set to the following hexadecimal values:

- X'01' - cannot open the user catalog for input
- X'02' - cannot open the user catalog for output
- X'04' - the scratch catalog member is unusable
- X'08' - the user catalog is unusable
- X'10' - user catalog input paging error
- X'11' - user catalog DSCB input paging error

Output: None

Modules Called:

READ/WRITE (CZCPE) -- To read or write a record in SYSSVCT.

READWRIT (CZCEM) -- To read or write a DSCB.

VMA (CZCGA) -- To get work and buffer pages (GETMAIN).

ADDDSCB (CZCEK) -- To allocate a new DSCB for a user catalog.

INDEX (CZCFI) -- To create a userid SBLOCK level when rebuilding the catalog.

GET SBLOCK (CZCFG) -- To set up the scratch catalog buffer (entry point 3). --To do PUT/PUTX/STOW on the scratch catalog member (entry point 4).

OPEN (CZCLA) -- To open a user catalog DCB.

PUT (CZCOS) -- To copy the scratch catalog member to the user catalog. --To copy the user catalog to the scratch catalog.

CLOSE (CZCLB) -- To close the user catalog DCB.

DELCAT (CZCFD) -- To delete a catalog entry.

VMA (CZCGA) -- To free the buffer and work pages (FREEMAIN).

LOCATE (CZCFL) -- To verify data set names with public DSCBs.

ADDCAT (CZCFA) -- To update DSDs. --To catalog public data sets during rebuilding process.

RELEXPG (CZCEN) -- To free up data pages and DSCB slots.

GET (CZCOR) -- To copy the scratch catalog member to a work area. --To copy the user catalog into a buffer for copy to the scratch catalog.

USER PROMPTER (CZATJ) -- To issue diagnostics to the user.

PAT LOCK (CZCEJ) -- To lock or unlock a PAT when writing a DSCB.

SRCH SDST (CZCQE) -- To clean up the SDST entry when called during the open process.

FIND (CZCOJ) -- To lock a scratch catalog member.

ABEND (CZACP) -- To terminate processing when an error occurs.

Exits:
Normal - Return to caller if input code is X'04' (SCRATCHCAT is unusable)

Error - ABEND (CC1)

Operation: DSCB/CAT RECOVERY assumes upon entry that the user catalog member is locked in the scratch catalog. This lock prevents multiple users from attempting recovery of a user catalog.

The userid is used as a key to extract the entry for this user from SYSSVCT. The pointer to the user catalog (UCTDSCB) is checked for a nonzero pointer.

If a user catalog exists (UCTDSCB is not zero), the DSCB is read into CZCOZ by a call to READWRIT (CZCEM). The return code is tested, and if the data set name agrees, this DSCB is used to update the catalog. The return code from READWRIT is saved for later use. The DSCB is checked for the following:

1. VS organization
2. U-format records
3. Number of pages in DSCB = Number of page entries
4. LRECL = a multiple of 4096
5. A checksum error

If any of these checks fails, the DSCB is declared invalid, and the DSNAME field in the invalid DSCB, the DSCB pointers in UCTDSCB and the JFCB for this user catalog are all filled with zeros.

The DSCB is then written back to external storage by READWRIT. ADDDSCB is called to assign a new DSCB slot, and the pointers in the JFCB and UCTDSCB are updated. The user catalog is then rebuilt in SCRATCHCAT by scanning public storage for the user's data sets and calling ADDCAT2 with the CATVAM option to catalog them. Upon completion of the rebuild, SCRATCHCAT is used to create a new USERCAT.

If the tests on the original DSCB were successful, a GETMAIN is done for a buffer large enough to hold all pages of the user catalog. A SETXP is then issued to read these pages into the buffer and the first page is checked for an index level SBLOCK with the correct userid in the name field. If the userid is incorrect, the user catalog is bad and the procedure described above for a bad DSCB is followed.

If the userid in the first SBLOCK on the first page of the user catalog is correct, all catalog chains are searched for public DSDs. For this search, the following tests are made to ensure catalog integrity:

1. Forward and backward pointers for each SBLOCK are good.
2. Number of bytes used in each SBLOCK is correct.

The count in the above test is corrected if it is wrong. If the pointers are wrong, the user catalog is bad, and it is rebuilt from the public DSCBs. At the completion of the search and corrections, the buffer is written (by PUT) into the scratch catalog for the completion of the recovery procedure.

Following the copy to the scratch catalog, all DSCBs on public volumes are searched and as each data set for this user is found, a LOCATE is done. If the data set does not exist in the catalog, it is cataloged by a call to ADDCAT2 with the CATVAM option specified. A dummy JFCB is used for this call with the data set name moved into it from the DSCB. If the data set does exist in the catalog, the DSCB pointer is compared to the address of the DSCB on public storage. If the pointers agree, the DSCB search is continued. If the DSCB pointers do not agree, the DSCB pointed to by the catalog is checked and, if it is a valid DSCB, the DSCB search is continued. If it is an invalid DSCB, the DSCB just located is checked: if it is valid, a pointer to it is put in the catalog; if it is not valid, the data set is erased. At the completion of the DSCB search, the scratch catalog member is again searched for public DSDs and any DSDs that are not marked as public are deleted from the scratch catalog by a call to DELCAT. At the completion of the scratch catalog search, the member is copied to the user catalog as described in the CATFLUSH module description.

If UCTDSCB is zero, public storage is searched for a USERCAT DSCB. If one is found, both the DSCB and the USERCAT are validated as described above. If a USERCAT DSCB is not found, the scratch catalog is rebuilt.

SECTION 2: EXTERNAL STORAGE ALLOCATION

External storage allocation (ESA) includes those service routines that allocate storage from direct access volumes designated as external volumes. External volumes are those volumes of secondary storage used for data, as opposed to volumes used as auxiliary storage. Auxiliary volumes are used for virtual storage (paging) and, as such, are controlled by auxiliary storage allocation (CEAIA).

EXTERNAL VOLUMES

The external storage allocation routines deal only with direct access volumes (IBM 2311 Disk Pack or IBM 2314 Disk Pack). All public storage volumes are assumed mounted.

Each volume in the system has a 6-byte identification associated with it. The volume identification is included in the volume label on each volume. When a device becomes accessible to the system, a unique 2-byte logical (symbolic) device address is associated with it; this logical address can then be used to refer to any volume mounted on that device. The logical device address is converted to a true physical address by the supervisor path finding mechanism whenever the volume is accessed. The symbolic device allocation table (SDAT) contains an entry for each on-line device. Within the entry is control data used by ESA routines to determine volume type, page availability information, Page Assignment Table (PAT) origin, PAT VMA for VAM volumes, and VTOC space available or gross space available for SAM volumes.

Each volume is identified by a volume label located on cylinder 0, track 0. The volume label points to the Page Assignment Table (PAT) for VAM volumes, which is used to indicate the current assignment status of each page on the volume. The PAT occupies some number of pages on the volume, the number dependent on device type (1 on the 2311 Disk Pack; 2 on the 2314 Disk Pack). The remaining pages can be allocated as Data Set Control Block (DSCB) pages or data pages. For SAM volumes the volume label points to the VTOC which is itself variable in length but composed of fixed length (140 byte) records (DSCBs).

SAM volumes use format-0, -1, -4, -5, -A, -B, and -C DSCBs. VAM volumes use format-E and -F DSCBs. The DSCB formats are described in detail in Appendix A.

There are two types of external volumes, those containing physical sequential data sets (SAM volumes) and those containing virtual storage data sets (VAM volumes). Each external volume may contain either, but not both, types on the same volume. VAM volumes are characterized by the fact that they are formatted into page-sized records and space is maintained and allocated in page-oriented extents. SAM volumes are not pre-formatted and can be used for interchange with Operating System/360. Space is maintained in terms of tracks and cylinders and can be allocated in increments of either tracks or cylinders.

Utility programs are provided with TSS/360 for volume initialization. Initialization of volumes containing physical sequential data sets is described in the publication IBM System/360 Time Sharing System: Independent Utilities, GC28-2038. Initialization of volumes containing virtual storage data sets differs from those containing physical sequential data sets in that VAM volumes are formatted into page-sized segments (to be consistent with page number conversion algorithms in the resident monitor) and the PAT table is then formatted and initialized. The PAT table will be pointed to by the Volume Label, and will contain one one-byte entry for each page on the volume.

PUBLIC AND PRIVATE VOLUMES

All external volumes can be classified as either public or private. Space is not allocated on a private volume unless a user explicitly declares in a DDEF command or DDEF macro instruction that he wants space on that volume. If the user does not declare a specific volume, space will be allocated on a public volume. All public volumes are VAM volumes, and the system specifies the storage device for the user.

DUPLEXING CAPABILITY FOR USER DATA SETS

Through the DUPOPEN command, the user has the capability of duplicating his data set on two separate physical volumes. This function is transparent to the user and is available only for public VAM data sets. The external storage required is exactly double for this duplicating facility, and the time required for data output is approximately doubled. The routines that

operate on VAM data sets must be aware of the possibility of the duplicate set.

Note: If either copy of a duplexed data set is changed independently of the other, duplexing is invalidated in a manner which is transparent to the duplexing mechanism, and may cause false recoveries.

SAM VOLUME PROCESSING

ALLOCATE is called to inspect the job file control block (JFCB) for the size of the initial allocation. The specification will be in terms of tracks, cylinders, or records, and only from a single volume having enough space for the entire amount of requested storage. ALLOCATE returns the appropriate code if the request cannot be satisfied, and updates the SDAT and volume fields of the JFCB.

ALLOCATE calls SAMSEARCH to find the extents for physical sequential data sets. During the scan, SAMSEARCH creates a push-down list of five extents smaller than the request. If an extent equal to the request is found, the scan is terminated and the extent allocated. If an extent equal to the request is not found, the smallest extent larger than the request is allocated. If an extent equal to or greater than the request is not found, space is allocated from the extents in the push-down list. Up to five separate contiguous extents as necessary are combined until the request is satisfied. Requests must be made for integral numbers of tracks or cylinders.

When the access method routine detects that extent limits have been reached, and additional space is required for a data set, EXTEND is used to obtain additional storage. The secondary allocation field of the JFCB is used to determine the amount of storage required. EXTEND always makes the total requested allocation from one volume. If there is not sufficient space available, EXTEND links to the calling program (end of volume - EOV) for label processing and the mounting of a new volume. EOV returns to EXTEND for the allocation from a new volume.

When a data set is closed, the CLOSE routine checks the space release flag in the JFCB to determine whether unused space is to be returned (that is, made available for reallocation, and made unavailable to the data set from which it is released). GIVBKS is called to merge the returned extents into the list of extents in the DADSM-DSCBs of the proper volume. Adjacent extents are combined whenever possible. If any extents are given back, the DADSM-DSCBs are updated.

To return the extents to the available space on the volume, MERGESAM is called by GIVBKS. It is MERGESAM that adds the returned extents to the list of extents in the DADSM-DSCBs, maintaining the order by location and combining extents whenever possible.

When the user commands the system to erase a data set, or when any data set remains on a public volume at the end of a task, the SCRATCH routine is used to remove it from the volume or volumes on which it resides. All the allocated extents are merged back into the available extents on the volume(s) and the data set DSCBs are over-written with zeros to indicate that the extents are available for reallocation.

The various SAM routines call the OBTAIN/RETAIN routine for reading and writing DSCBs. The RENAME routine can be used to change the FQN in the key field of a format-1 DSCB.

VAM VOLUME PROCESSING

VAM volumes may be either public or private. FINDEXPG is called to allocate the number of pages specified in the allocation fields of the JFCB. The new DSCB will be constructed using the list of external page entries returned from FINDEXPG.

To find and commit an unassigned DSCB slot, ADDDSCB is called. For a format-E DSCB, ADDDSCB will call VOLSRCH and will search the PAT of the returned RVN to find an unused DSCB or a data page which can be assigned as a DSCB page. For a format-F DSCB, ADDDSCB will try to assign it from the page or volume on which the format-E DSCB resides. Failing this, it will call VOLSRCH as for a format-E DSCB. RELEXPG is used to return data pages and DSCBs for subsequent reallocation.

DSCBREC is called to recover from a checksum error. If the data set has been opened, the bad format-E DSCB is replaced by a new one. This is because the call is assumed to have come from WRITDSCB after having found a bad format-E DSCB. All the data set pointers are present in the RESTBL, and the DSCB chain is about to be updated anyway. If the data set has not been opened and there is no duplexed copy, the data set is deleted from the catalog and all DSCBs associated are released. If the data set is not open, but is duplexed, recovery is made from the duplexed copy.

WRITDSCB updates and writes the DSCBs to external storage, using an in-line subroutine to obtain the DSCBs to be updated.

When private VAM volumes require mounting, the VAMINIT routine is used to read the volume label record, and set up the SDAT entry.

ROUTINES USED WITH SAM FORMAT VOLUMES

ALLOCATE (CZCEA)

ALLOCATE is a reentrant, nonrecursive, privileged routine residing in virtual storage. This routine is called by DATADEF to provide the initial allocation of direct access external storage for new output data sets. ALLOCATE finds a private volume in the JFCB. DSCBs describing the extents allocated are written and the VTOC DSCB, JFCB, and SDAT entry updated. (See Chart BA.)

Entry Points:
CZCEA1 -- Normal entry via Type I linkage.

CZCEA2 -- Called by ABEND for resetting interlocks.

Input: Register 1 contains a pointer to a one-word parameter list:

Word 1 Address of the JFCB

Output: The JFCB, VTOC, and SDAT are updated.

Restrictions:

1. Allocation is from one volume only.

2. There is a maximum of five extents allocated to one SAM data set.

Modules Called:
OBTAIN/RETAIN (CZCFO) -- Read/Write DSCBs

SAMSEARCH (CZCEC) -- Assigns extents

PAIR (CZCACS) -- Set and delete ABEND table entry

SYSER (CEAIS) -- Minor system error

ABEND (CZACP) -- End task and return control to terminal

Exits:
Normal - Register 15 contains 00.

Error - Register 15 contains one of the following codes:

04 - No space found
08 - Unrecoverable I/O error
0C - Duplicate DS name found
10 - Volume not formatted properly

SYSER and ABEND occur when the JFCB does not indicate any mounted volume.

Operation: ALLOCATE searches the volume fields of the JFCB to find the last mounted volume. If none are found, SYSER is called, followed by ABEND with the following message to the user:

NO MOUNTED VOLUME INDICATED BY JFCB

If one is found it is checked to be sure it is a SAM formatted volume on a private device. If it is on a public device, SYSER is called and return is made with the return code set to '04'. If the volume is not SAM formatted, a return is made with the return code set to '10'. If these checks are successful, the VTOC is locked and the gross space and hole count fields of the SDAT are examined. Return is made with a code of '04' if there is not enough gross space on the volume. If there is space enough, OBTAIN is called to read the DSCB with the same data set name on the volume. If such a DSCB is found, through a SEARCH KEY EQUAL on DSNAME, a duplicate data set name exists, and return is made with a code of '0C'. If none is found, ALLOCATE calls SAMSEARCH to allocate space for the DSCB. OBTAIN then reads in two format-0 (i.e., unused) DSCBs to become the data set DSCBs (format-1), sets up all fixed DSCB fields and fills in the extents just allocated. The original data set flag is set in the JFCB; the VTOC is updated, the lock bytes reset, and return is made with a code of '00'. An unrecoverable I/O error will cause the program to return with a code of '08'.

SAMSEARCH (CZCEC)

SAMSEARCH is called by either ALLOCATE or EXTEND to search the DADSM-DSCBs for available space to fill a request. The DADSM-DSCBs are then updated to reflect the allocation. (See Chart BB.)

Entry Point: CZCEC1

Input: General Register 1 contains a pointer to the following parameter list:

Word 1 Address of either the primary allocation field (TDTSP0) or the secondary allocation field (TDTSP2) of the JFCB

Word 2 Address of the VTOC-DSCB

Word 3 Address of the SDAT entry

Word 4 Address of a 12-word output area. The first byte of the area contains a flag indicating the type of request.

Output: General register 1 contains a pointer to the input parameter list; the list is unchanged, but the work area

pointed to by Word 4 now describes the extents allocated.

If a standard user label (SUL) track was requested, the output area has the following format:

Word 1	Count of extents allocated
Word 2	CCHH of SUL track
Word 3	CCHH of lower limit of first extent
Word 4	CCHH of upper limit of first extent
Words	CCH of lower and upper limits of other extents

If no SUL track was requested, the output area has the following format:

Word 1	Count of extents allocated
Word 2	CCHH of lower limit of first extent
Word 3	CCHH of upper limit of first extent
Words 4-11	CCHH of lower and upper limits of other extents
Word 12	Unused

Modules Called:
OBTAIN/RETAIN (CZCFO) -- read/write format-5 DADSM-DSCBs from/to external volume

WTO macro (CZABQ) -- write error message to system operator

SYSER (CEAIS) -- minor system error

ABEND (CZACP) -- force abnormal end of task

Exits:
Normal - registers 15 contain 0, registers 0-14 are restored

Error - register 15 contains 04, meaning request cannot be satisfied; registers 0-14 are restored

Operation: The size of the requested area is extracted from the JFCB. On a request for track allocation, the number of tracks requested is incremented by 1 if a standard user labeling (SUL) track is requested; if a SUL track is requested the "SUL added" flag is set.

The SEARCH routine (described below) then finds the extent or extents to satisfy the request. The COMPUT routine (described below) examines the extents to see if they fill the request, sets appropriate pages, and updates the DSCB. UPDATE (described below) updates the DADSM-DSCBs to reflect the allocation of extents. Return is then made to the calling routine.

SEARCH SUBROUTINE: OBTAIN reads the DADSM-DSCBs, one at a time, into a work area. UNPACK (described later) examines the extents and computes the size of the request. The request can be in terms of tracks (in which case the allocation can begin at any available track) or in terms of cylinders (in which case only full cylinders can be allocated). The request can be satisfied in one of three ways:

- An extent equal to the request.
- An extent larger than the request.
- A combination of extents, each of which is smaller than the request.

If an extent equal to the request is found, the extent is placed in COMPUT's work area, and, if no SUL track is requested, exit is made. If, however, a SUL track is requested, the extents are searched for a surplus track to be used as a SUL track. If the DSCB is chained, OBTAIN reads in the continuation DSCBs and they are searched for an available track. If a SUL track is found, a successful return is made. If no such track is found, a no-space-available return is made.

If an extent greater than the request is found, it is saved until an extent equal to the request, or a smaller extent that is larger than the request is found. If the DSCB is chained, all continuation DSCBs in the chain are searched. If an extent equal to the request is found, the procedure described in the preceding paragraph is followed. If no extent equal to the request is found during the search, the request is satisfied from the smallest extent that is larger than the request. The excess space is subtracted from the extent and the result is entered into the DSCB as an update to the old extent. If a SUL track is requested, the procedure described in the previous paragraph is followed.

During the search of the DSCBs, a push-down list of extents smaller than the request is accumulated. The list has five entries, and only extents larger than the smallest extent in the list are added. When a new extent is added, the smallest is deleted from the list. This list is abandoned as soon as an extent equal to or larger than the request is found. If no such extent is found, space is allocated from the extents in this list, combining the smaller extents (with a maximum of five) for the allocation.

COMPUT SUBROUTINE: COMPUT determines if an extent can fill the request and sets the appropriate flag accordingly. It also determines if the entire extent is allocated and updates the DSCB accordingly.

If a SUL track is requested, the output parameter list pointer is incremented to include the SUL track. The requested allocation can be in terms of tracks or cylinders.

- If a cylinder request also includes an SUL request, COMPUT determines if there are any available tracks preceding the cylinder. If there are, the SUL track is made the first track of the extent. If there are no preceding tracks, the last track on the cylinder is used as the SUL track. If the entire extent was allocated, its entry in the DSCB is flagged for deletion by UPDATE; if only part of the extent was allocated, the DSCB entry is flagged for updating (by UPDATE) to reflect the new limits of the extent. If space in addition to a SUL track was allocated, COMPUT places the upper and lower limit CCHH of the extent in the output parameter list, and checks the input parameter list for additional extent requests.

- If a track request also includes a SUL track, the first available track in the extent is used as the SUL track and its CCHH is placed in the output parameter list. If the entire extent was allocated, the extent entry is flagged for deletion. If the extent allocated was only a SUL track, COMPUT examines the next entry in the input list; if the extent included other tracks in addition to this SUL track, the upper and lower limit CCHHs are placed in the output list before examining the next input entry.

When all entries in the input list have been processed, a count of extents allocated is placed in the output list, and the gross available space indicator for the volume is updated. Exit is then made.

UPDATE SUBROUTINE: UPDATE updates the DADSM-DSCBs as extents are allocated. OBTAIN is called to read in the DSCBs if they are not already in storage. The CCHHR and chain address in the input list are moved to the output list, if needed; otherwise, they are saved.

If the extent in the DADSM-DSCB equals zero and UPDATE's work area list is not exhausted, SYSER is invoked. If the extent in the DADSM-DSCB is not zero, the available tracks in the DSCB extent is compared to the original request for track allocation of the entry to UPDATE. If the DSCB extent available tracks is greater, SYSER is invoked; if it is lesser, PACK is called to move the DSCB extent to the output list. If there are more extents in this DSCB, the DSB pointer is incremented, and the next extent in the DSCB is examined, as above. If there are no more extents in the DSCB, OBTAIN is called to read in the next DSCB, and its extents are examined as previously described.

If, however, the number of available tracks in the DSCB extent is equal to the original request of the UPDATE entry, the following procedure is used.

PACK is used to move the UPDATE extent to the output work area, if it is not flagged "delete extent." If the next entry is a new entry to be added to the DSCB, PACK is called to move it to the output work areas; otherwise, the pointer to the next UPDATE entry is incremented. If there are more extents in this DSCB or if there are no more UPDATE entries, the DSCB pointer is incremented, and the next extent in the DSCB is examined, as described previously. If there are no more extents in the DSCB, OBTAIN is called to read in the next DSCB, and its extents are examined.

If the extent in the DADSM-DSCB equals zero and the UPDATE work area list finally becomes zero (all DSCBs have been examined), the output area is zeroed and the hole count in the VTOC-DSCB is incremented by one, if the DSCB is not the first in the chain. RETAIN is then used to write the DADSM-DSCB.

If a hardware error occurs, the WTO macro instruction is issued to inform the operator; registers are restored, a return code of 8 is set, and return is made. For all the errors, SYSER is invoked, the WTO macro instruction is issued to inform the operator, registers are restored, a return code of 04 is set, and return is made.

PACK SUBROUTINE: If the output-DSCB-full flag is set, OBTAIN is called to get the CCHHR for available continuation DSCBs, if any. RETAIN is called to write the DADSM-DSCB. A CCHHR, if one was saved, is used as the address of a new output DSCB. Allocated extents are moved to the output DSCB. If there are more extent slots in the output DSCB, PACK moves its pointer to the next available slot and exits. If there are no extent slots in the output DSCB, the output-DSCB-full flag is set, and exit is made.

UNPACK SUBROUTINE: If the next extent in the DSCB is 0, the DSCB is scanned for a chain address; if there is none, normal exit is made; if there is one, an error exit is taken.

If the next DSCB extent is not 0, it is put in a work area; the count of extents in the DSCB is incremented by one. The size of the extent in tracks is computed or set equal to the cylinder count, depending on the type of request. If there is room for more extents in the DSCB, this process is repeated. When all the extent fields of the DSCB are filled, exit is made.

SCRATCH (CZCES)

SCRATCH deletes data set DSCBs on all volumes of a specified data set and assimilates the external storage back into the available space (the DADSM) on the volume. (See Chart BC.)

Entry Point: CZCES1

Input: Register 1 contains a pointer to a word containing the address of a JFCB.

Output: None

Assumptions: At least one of the volumes indicated in the JFCB is mounted upon entry to SCRATCH.

Modules Called:
BUMP (CZCAB) -- dismount and mount external volume on same device

OBTAIN/RETAIN (CZCFO) -- read/write DSCBs from/to external volume

MERGESAM (CZCEE) -- merge physical sequential data set extents with DADSM extents

VMA (CZCGA) -- allocate and release virtual storage

WTO macro (CZABQ) -- write error message to system operator

SYSER (CEAIS) -- minor system error

ABEND (CZACP) -- force abnormal end of task

PAIR (CZACS) -- set and delete ABEND Table Entry

Exits:
Normal - register 15 contains 0; registers 0-14 are restored.

Error - register 15 contains 04, meaning SCRATCH was unsuccessful on one or more volumes as indicated in JFCB volume fields, as follows:

TDTV8 = 0; TDTV9 = 0 - successful

TDTV8 = 0; TDTV9 = 1 - error due to system problems

TDTV8 = 1; TDTV9 = 1 - data set not found

Registers 0-14 are restored.

Exit to ABEND if a mounted volume cannot be found, or if an error is encountered by BUMP.

Operation: The first volume field in the JFCB is examined. If a JFCB chain field or other than a data set volume is selected, the next volume field in the JFCB is selected until a data set volume is found. If the selected volume is mounted but cannot be found, the next volume field in the JFCB is selected. If a mounted data set volume is found but has already been processed, BUMP is called to mount the next volume of the data set. If a mounted volume is not found, the mounted-volume-found indicator is set, the relative volume number of the volume that was found is saved, and the parameter list for BUMP is initialized.

If the VTOC is locked, time-slice end is invoked until such time as the VTOC becomes available. The VTOC lock byte is then set, and OBTAIN is called to read in the first data set DSCB (format-1) and the VTOC-DSCB. GETMAIN is called to get storage for the extent list, and the list is constructed from the format-1 DSCB. The DSCB is then zeroed out, and RETAIN is called to write it back as a zero record; the VTOC hole count is incremented by one. If the data set DSCB is chained, the continuation DSCBs (format-3 or -0) are read in by OBTAIN, zeroed out, and written back as zero records by RETAIN. The VTOC hole count is incremented by one for each such DSCB returned.

MERGESAM is then called to return the free extents to the available space on the volume. FREEMAIN then releases the storage used for the extent list. RETAIN writes back the updated VTOC-DSCB, and the hole count in the VTOC and the gross available space indicator in the SDAT are updated. The VTOC lock byte is reset and the next volume field in the JFCB is examined. This process is repeated for each volume on which the data set resides.

When all volumes have been processed, SCRATCH returns with a return code as indicated under Exits.

EXTEND (CZCEX)

EXTEND is the external storage allocation routine that is called when additional space on a direct access volume is required for a data set. EXTEND makes the allocation and updates the VTOC, JFCB, and SDAT.

This routine is called by SAM EOV when an end-of-volume condition is encountered and additional direct access space must be allocated. (See Chart BD.)

Entry Points:
CZCEX1, normal entry point.

CZCEX2, used only by SAM EOV when calling EXTEND after having a new private volume mounted.

Input: Register 1 contains a pointer to a two-word parameter list.

Word 1 Address of JFCB

Word 2 Address of a word in which EXTEND places output address. If this word has a value of 1, FEOV has been requested on current volume.

Output: The word pointed to by parameter two contains an address pointing to the first DSCB containing the new extents, followed by a doubleword containing the CCHHR address of the DSCB. If extents are contained in a second DSCB, the word following the doubleword contains the address of the DSCB followed by its CCHHR. If no such DSCB exists, the address is zero.

Restrictions: Each allocation is made from one volume only; a maximum of 5 extents are allocated to a physical sequential data set.

Modules Called:
OBTAIN/RETAIN (CZCFO) -- read/write DSCBs

SAMSEARCH (CZCFC) -- assign extents to data set

MTREQ (CZCAA) -- allocate additional private volume

SYSER (CEAIS) -- minor system error

ABEND (CZACP) -- force abnormal task end

PAIR (CZACS) -- ABEND deinterlock routine

Exits:
Normal - Register 15 contains 00.
Register 15 contains 04 - New Volume.

Error - Register 15 contains 08 - No space available.

Exit to ABEND if either of these conditions exists:

1. No mounted volumes in JFCB.
2. Unable to obtain Format-1 DSCB from data set volume.

Operation: The internal subroutine GET is used to retrieve the last mounted volume in the JFCB. The TEST subroutine is used to determine if there is sufficient space on the volume. It checks the SDAT hole count, the gross space field, and the VTOC lock byte. If the volume satisfies the request, TEST OBTAINs the VTOC-DSCB and calls SAMSEARCH to allocate space for the request. OBTAIN reads in a data set DSCB (Format-1) and 1WRITE enters the newly allocated extents into it. The 1WRITE internal subroutine is used to write extents in a DSCB when space allocation is from an old volume. 2WRITE is used when space allocation is from a new volume (see below). The VTOC-DSCB, JFCB, and SDAT are updated, and a successful return is taken.

If the last volume in the JFCB list does not contain enough available space, EXTEND will return to the calling program (SAMEOV) with a return code of 08. SAM EOV then has a new volume mounted. EXTEND is then entered at its secondary entry point (CZCEX2); the newly mounted volume is then the last volume in the JFCB list. The TEST internal subroutine is used to determine if the volume is suitable. If the volume does not contain sufficient space, return is made with a code of 08; if sufficient space is available, the 2WRITE subroutine sets up a new Format-1 DSCB and writes the extents. Continuation DSCBs, if needed, are read and written using OBTAIN and RETAIN respectively. The VTOC-DSCB, JFCB, and SDAT are updated and return is made with a code of 0.

GIVBKS -- Give Back SAM Storage (CZCEG)

GIVBKS returns unused external storage from physical sequential data sets to ESA control and deletes the references to the storage from the format-1 and -3 data set DSCBs. GIVBKS is called only by SAM CLOSE. (See Chart BE.)

Entry Point: CZCEG1

Input: General Register 1 contains a pointer to this list:

Word 1 Address of the JFCB

Word 2 Address of extent list

The extent list contains a 4-word entry for each extent being returned; there can be a maximum of 256 entries. The entry has this format:

Word 1	X'80'	extent sequence number	unused	
	byte 1	byte 2	byte 3	byte 4
Word 2	Lower cylinder and track CCHH of the extent			
Word 3	upper cylinder and track CCHH of the extent			
Word 4	unused		symbolic device address of the volume	
	byte 1	byte 2		

Notes: The flag in byte 1 of Word 1 denotes the last extent in the list; this byte contains binary 0, for all other entries. The symbolic device address of the volume in bytes 3 and 4 of Word 4 is only in the first extent of the list. These bytes are unused in all other entries, since extents can be returned only for one volume.

Output: None

Restrictions: Extents can be returned for one volume only.

Modules Called:
OBTAIN/RETAIN (CZCFO) -- read/write formats -1, -3, and -4 DSCBs from/to external volume

MERGESAM (CZCEE) -- merge physical sequential data set extents with DADSM extents

VMA (CZCGA) -- allocate and release virtual storage

SYSER (CEAIS) -- minor system error

ABEND (CZACP) -- force abnormal end of task

Exits:
Normal - registers 0-14 are restored: register 15 contains 0, meaning GIVBKS successful or 04, meaning GIVBKS successful and the entire prime data set DSCB (format-1) has been zeroed.

Error - registers 0-14 are restored, register 15 contains 08, meaning GIVBKS unsuccessful.

Operation: A JFCB and the symbolic device address are passed as input; from this, GIVBKS finds the volume on which space is to be returned. GIVBKS obtains the SDAT pointer, sets the VTOC lock byte, and calls OBTAIN to read the VTOC-DSCB and the format -1 DSCB for the data set. The amount of space required for reading continuation DSCBs and constructing the extent list for MERGESAM is computed from the number-of-extents field of the format-1 DSCB. This figure must be increased by one if the first extent is a label track, since that extent is not included in the count. A list of extents is then constructed; the manner in which this is done depends on the number of extents being returned.

More Than Three Extents: If there are more than three extents described, the space required for format-3 DSCBs and the MERGESAM list is computed as follows:

Every record read into the work area is preceded by a marker doubleword to identify end-of-record condition during the scan process. Consequently, 148 bytes of storage are required for every record (144 for the DSCB and four for the marker), containing up to thirteen extents, and eight bytes are required for every extent to construct the MERGESAM parameter list. The space-requirement computation is therefore:

$$\left\{\left[\frac{N+12}{13}\right]\times 148 + 8N\right\}$$

bytes, where N is the count of extents, adjusted for label track if necessary.

This space is obtained by use of GETMAIN.

The marker words are moved into the area. OBTAIN is called to bring in the format-3 DSCBs; a continuation character indicates that there are more to be read in.

The last extent in the last format-3 DSCB is located; then the last extent in the list is found. The index is set to the end of MERGESAM's list, to enter the DSCB being returned.

Not More Than Three Extents: If there are not more than three extents, MERGESAM's list index is set to the end of the PSECT work area. The last valid extent in the format-1 DSCB is found, the number of extents returned is counted, and an index is set to the end of the input list. The last extent in the last format-3 DSCB is located; then the last extent in the list is found. The index is then set to the end of MERGESAM's list to enter the DSCB being returned.

Each extent in the DSCB is zeroed out, and the count of extents still to be returned is decremented by one for each extent. This process is repeated until the count of extents to be returned is zero (i.e., there are no more extents to return), or until the lower level CCHH of the extent being returned does not compare with that of the extent in the DSCB (meaning that part of the extent was used and

the entry in the DSCB must be changed to reflect this condition). The number of extents in the format-1 DSCB is adjusted by the number being returned. However, when only part of an extent is returned, the resultant number of extents must be incremented by one.

When a continuation record (a format-3 DSCB) is zeroed out because of returned extents, RETAIN is called to write it back as a zero record, and the hole count in the VTOC-DSCB is incremented by one. If all extents for a data s et on a particular volume are returned, the format-1 DSCB is also zeroed; RETAIN is again used to write it back as a format-0 DSCB, and the hole count in the VTOC-DSCB is incremented by one. The format-1 DSCB is updated, and RETAIN is called to write it back.

After all the data set DSCBs (formats-1 and -3) have been adjusted, MERGESAM is called to merge the returned extents with the available space on the volume. RETAIN is called to rewrite the VTOC-DSCB. The hole count indicator and gross available space field in the SDAT are adjusted according to the values in the VTOC-DSCB. The VTOC lock-byte is reset, and FREEMAIN is called to release work areas. Return is then made to SAM CLOSE.

MERGESAM (CZCEE)

MERGESAM returns extents from physical sequential data sets passed to it by SCRATCH or GIVBKS and merges them with the DADSM extents on their volume. (See Chart BF.)

Entry Point: CZCEE1

Input: Register 1 contains a pointer to the following parameter list:

Word 1 Address of an SDAT entry

Word 2 Pointer to a list of data set extents

Word 3 Address of the VTOC DSCB

Word 4 Pointer to HW containing number of extents

Output: None

Modules Called:
OBTAIN/RETAIN (CZCFO) -- read/write format-5 DADSM-DSCBs from/to external volumes

WTO macro (CZABQ) -- write error messages to system operator

SYSER (CEAIS) -- minor system error

Exits:
Normal - register 15 contains 0; registers 0-14 are restored

Error - None

Operation: The list of extents to be merged with the DADSM extents is first sorted; contiguous extents are merged and the number of extents is adjusted accordingly. The extents are put into DADSM form. OBTAIN reads the DADSM-DSCBs until the first extent is within the range of the DADSM extents in that DADSM-DSCB. If the extent cannot be merged with a DADSM extent, it is inserted, and following extents are pushed down to make room. If this causes the DADSM extents to overflow the DADSM-DSCB, OBTAIN reads an available DSCB to create a new DADSM-DSCB chained to the other with the overflow extent inserted. The hole count (available DSCB records) in the VTOC-DSCB is decremented by one. If the list extent can be merged with one of the DADSM extents, the DADSM extent is modified to reflect the list extent also; no other action is required in this case. If, however, the list extent can be merged with two of the DADSM extents, the three extents are merged into one DADSM extent and the following extents are shifted up to fill the hole created. The last vacated DADSM extent field is zeroed out. If, in this process a DADSM-DSCB becomes empty, its key and data are zeroed out (making it a zero DSCB), and RETAIN writes it back; the hole count in the VTOC-DSCB is incremented by one. This procedure is repeated for all extents in the list.

When the list of extents to be returned is exhausted, the holes, if any, in the DADSM-DSCB are filled. OBTAIN is called toread in DSCBs and the DADSM extents are pushed up. RETAIN writes out the DSCBs after the holes have been filled. This process continues until either there are no more holes or until all DADSM-DSCBs have been processed.

Should a DSCB become vacant as a result of the push-up process, it is zeroed out, and written back as a zero DSCB. When the extents in all the DADSM-DSCBs have been processed, successful return is made.

OBTAIN/RETAIN (CZCFO)

OBTAIN/RETAIN is a reenterable, nonrecursive, privileged routine residing in virtual storage. This routine is used for SAM format DSCBs with the exception of the option for reading or writing volume labels, which can be used with either VAM or SAM.

OBTAIN reads the VTOC, obtains virtual storage, and builds an IORCB. A channel

program is constructed in the IORCB according to the type of OBTAIN requested. When the IORCB is completed, an IOCAL is issued followed by an internal check routine (AWAIT if I/O is not complete). A posting routine checks for errors when I/O is completed and links to SYSER if necessary. If successful, the data is moved from the IORCB buffer to the user's input area.

RETAIN writes one or more DSCBs, volume labels, or end-of-file markers to specified addresses. Virtual storage is assigned and an IORCB constructed. A channel program is then developed and an IOCAL is executed. When I/O is complete, a posting routine checks for errors and links to SYSER if necessary. (See Chart BG.)

Entry Points:
CZCF01 - OBTAIN
CZCFO2 - RETAIN
CZCFR1 - RETAIN -- Same entry point as CZCF02.

Input: General register 1 contains a pointer to the parameter list:

Word 1 A pointer to the symbolic device address entry table (SDAT) for the volume containing the VTOC or data set. This field is a 32-bit virtual storage address.

Word 2 A pointer to a packed word that is arranged as follows:

bits 0-7 - Type of OBTAIN request desired. The type is designated by a binary number as indicated in Table 1. The type is designated by a binary number as indicated in the 'RETAIN REQUEST' section.

bits 8-15 - used type of RETAIN request.

bits 16-31 - count of RETAIN requests; otherwise - binary zeros.

Word 3 A pointer to a field containing a data set key or a label key when requesting a Type 1, 2, or 5 OBTAIN. When this field does not apply to the function requested, it must be defined but can be left blank. The data set key is assumed to be 44 bytes in length, and a label key is assumed to be 4 bytes in length.

Word 4 A pointer to a CCHHR when performing direct reads for formats 3, 5, B, and D DSCBs, or when writing DSCBs. When reading a label, this field will point to the CCHH of the track upon which the label resides.

Note: When writing records, fields 4 and 5 are repeated as many times as the count in word 2.

Word 5 A pointer to the input area for the OBTAIN request being made or the address of the record to be written.

Type	Length
0	140 bytes
1	101 bytes
2	244 bytes
3	5 bytes
4	10 bytes
5	80 bytes

or

A pointer to the input area for the RETAIN request being made.

Type	Length
0	140 bytes
1	84 bytes
2	user specified

Note: OBTAIN and RETAIN use identical parameter lists.

Output: Register 1 points to the input parameter list.

Restrictions:
1. The writing of DSCBs, labels, and end-of-file marks cannot be intermixed.
2. OBTAIN/RETAIN does not set the interlock byte in the symbolic device allocation table (SDAT). To prevent concurrent references to a particular VTOC, the user must test and set the lock byte.

Modules Called:
IOCAL (CEAH16) -- execute request for I/O.

AWAIT (CEAP7) -- wait for I/O completion.

VMA (CZCGA) -- GETMAIN for the virtual storage used to construct the IORCB.

RESET (CEAAH) -- re-enable an I/O device.

SYSER (CEAIS) -- entered when request type or parameter is invalid.

ERRORRETRY (CZCRH) -- direct access error retries.

VMSDR (CZCRY) -- VM statistical data recording.

Exits:
Normal - register 15 contains 00.
Error - register 15 contains one of the following codes:

04 - write error (RETAIN)
04 - DSCB record not found (OBTAIN)
08 - hardware error condition (OBTAIN)
08 - error in input parameter list (RETAIN)
0C - unit exception (OBTAIN)
10 - intervention required (OBTAIN type 5)

Operation: For both OBTAIN and RETAIN requests, upon entry a data control block (DCB), a data extent block (DEB), and an event control block (ECB) are constructed in order to maintain compatible linkage with common system hardware error routines and the task monitor. A page of virtual storage is allocated through GETMAIN, and an initialized copy of an input/output request control block (IORCB) is moved into a reserved section of the page.

OBTAIN REQUEST

This copy of the IORCB is then updated to perform the requested read function. The location of the symbolic device address and the location of the VTOC are computed, fetched, and placed in the IORCB. A channel program is developed within the IORCB as required to perform one of the types of request shown in Figure 4.

Type	Function	Input Area	Description
0	Direct Reference DSCB	140 bytes	This request is initiated to read Format 3, 4, 5, B, and C DSCBs where the logical address of the record is known from a previous operation such as Type 2 (below). The address of the read is provided to the OBTAIN routine in the form CCHHR. A search for an equal ID is performed. When the ID is found, the key and data fields are read (140 bytes).
1	Search for DSCB key and read count	101 bytes	This request is initiated to read type 1 and A DSCBs. An in-channel search is performed on a key, which is the dsname of the data set whose DSCB is being searched for. When an equal key is found, the catalog data field (96 bytes) is read into the IORCB buffer area. The count field (5 bytes) is then read into the buffer area to occupy the byte of the data field. The count field is decremented by one to adjust it to the correct address of the data field on files.
2	Search for DSCB -- Read Count and Read DSCB	244 bytes	This request is the same as type 1 above, except that the VTOC-DSCB is also read and placed in the area contiguous to the right-hand byte of the count field starting on the next fullword boundary.
3	Search for one available DSCB	5 bytes	Unassigned DSCBs contain a field of binary zeros. This request causes an in-channel search on a field of binary zeros. The count field is read in the form CCHHR and is placed in the input area.
4	Search for two available DSCBs	10 bytes	This request performs the same functions as types above except that it searches for two available DSCBs. The first address is placed in the first five bytes of the input area; the second address is placed in the second five bytes.
5	Volume/User label read	80 bytes	This request initiates a track search for a data set or volume label key. When an equal is found on the key, the data field is read. If an end-of-file is detected on the read, a code is placed in register 15 and control is returned to the user. This request can be used for either VAM or SAM data sets.

Figure 4. Types of OBTAIN Requests

The initial DADSM-DSCB (the DSCB used to define available extents) can be accessed directly by incrementing the record address of the VTOC.

After the IORCB has been built, the IOCAL SVC is executed, followed by the OBTAIN routine and an AWAIT. When I/O is complete, the task monitor links to OBTAIN posting, which checks for errors. If no errors occurred, posting moves the data from the IORCB buffer area (now located in the ISA) to the user's input area. If an error occurred, posting links to common system error routines. Posting, however, selects out end-of-cylinder and end-of-file for further processing by OBTAIN and the user. If the IORNP or the IORPG flag in the IORCB has been set by either the HOLD/DROP facility, paging error recovery procedures, or the Purge I/O facility to indicate that no path is available, OBTAIN posting will post the I/O action as incomplete and intercepted. When posting completes, it returns control to the user with an indicative code in register 15, or updates the IORCB for another search if the VTOC exceeds a cylinder and end-of-cylinder was detected.

RETAIN REQUEST

Types of RETAIN requests are:

Type	Function	Input Area	Description
0	WRITE DSCB	140 bytes	This request is initiated to write one or more DSCBs. Available location found by type-3 OBTAIN
1	WRITE LABEL	84 bytes	To write a user label or standard label on a volume -- key must be included
2	EOF markers	user specified	

When the initialized IORCB, which contains an appropriate channel program to write the DSCB or label, is moved into a reserved section of the page allocated through GETMAIN, the symbolic device address is obtained from the symbolic device address table (SDAT) and placed in the IORCB. The seek and search address is directed to constant fields in the IORCB that are filled in with data indicated by the input parameter list. The write is also addressed to a constant buffer area in the IORCB into which the DSCBs or labels are moved. Note that a type 1 RETAIN request writes an 84 byte label. When the IORCB has been built, the IOCAL SVC is executed, followed by the RETAIN CHECK routine which checks the flag in the DECB. The RETAIN routine branches to an AWAIT macro instruction if the I/O is not complete.

Upon completion of the I/O, the task monitor invokes the RETAIN posting routine to check for hardware errors. If no errors have occurred, posting places an appropriate code in the DECB and returns control to the task monitor which, in turn, returns control to RETAIN at the instruction immediately following the AWAIT. If an error is detected, posting links to the system error routine for retry procedures.

If the IORNP or the IORPG flag in the IORCB has been set by either the HOLD/DROP facility, paging error recovery procedures, or the Purge I/O facility to indicate that no path is available, RETAIN posting will post the I/O action as incomplete and intercepted. Whether or not the retry was successful, control is returned to RETAIN by posting. RETAIN checks the DECB for an unsuccessful write, and if one has occurred, the hexadecimal error code '04' is placed in register 16. The count of the number of RETAIN requests in the parameter area is set to indicate the number of writes completed. Thus, the record causing the error can be located by multiplying the count by eight and incrementing the address of the first CCHHR pointer by the result. Control is returned to the user upon detection of the first error.

If no error is indicated, the count of records to be written is checked for zero; if the count is zero, control is returned to the user. When the count is not zero, the next record specified in the parameter list is written. Although an indefinite number of records can be specified in the RETAIN macro instruction, the RETAIN routine writes them one at a time in order to facilitate error recovery procedures.

The OBTAIN/RETAIN routine does not set an interlock in the symbolic device allocation table (SDAT) to prevent concurrent reference to a particular VTOC. Interlock must be established prior to issuing an OBTAIN or RETAIN if it is desired to prevent concurrent access to a VTOC. The following error checks will be made:

1. Validity of request type
2. Reasonableness of parameters

An invalid request type or parameter will cause the System Error Routine to be invoked.

RENAME (CZCFZ)

RENAME is a reenterable, nonrecursive, privileged subroutine residing in virtual storage. It changes the fully qualified name in the key field of a format-1 DSCB for all volumes specified in the JFCB to the name specified by the calling program. (See Chart BH.)

Entry Point: CZCFZ1

Input: Register 1 contains a pointer to the following parameter list:

Word 1 Address of a JFCB

Word 2 Address of a 44-byte area containing the new data set name

Output: None

Assumption: At least one of the volumes indicated in the JFCB is mounted upon entry to this routine.

Modules Called:
OBTAIN/RETAIN (CZCFO) -- read/write format-1 DSCBs from/onto external volumes.

BUMP (CZCAB) -- dismount and mount external volumes on same device.

WTO (CZCABQ) -- write error message to system operator.

SYSER (CEAIS) -- minor system error.

ABEND (CZACP) -- force abnormal end of task.

Exits:
Normal - register 15 contains 00: registers 0-14 are restored.

Error - register 15 contains 04; RENAME was unsuccessful on one or more volumes as indicated by flags:

TDTV8 = 0 and TDTV9 = 0 - Successful.

TDTV8 = 0 and TDTV9 = 1 - error due to system problems.

TDTV8 = 1 and TDTV9 = 0 - data set not found.

TDTV8 = 1 and TDTV9 = 1 - new data set name already exists.

Registers 0-14 are restored.

Operation: This routine uses OBTAIN to search the VTOC to determine if there is already a DSCB with the new data name and to retrieve the DSCB with the data set name to be changed. If a DSCB with the new data set name is encountered, an error return is made indicating that the name is not unique in the volume.

The routine performs the renaming for each volume on which the data set resides as specified in the JFCB. If any of the volumes is not mounted, the BUMP routine (CZCAB) is called to dismount a volume on which RENAME has already been performed and to mount the unmounted volume on that device.

If RENAME cannot perform its service on a volume, an indicator in the JFCB is set to pass the information back to the calling program. There are two bits within the volume flags of the JFCB defined as follows:

(Fields TDTV8 and TDTV9 in the system table CHATDT)

00 -- successful

01 -- unsuccessful due to system problems

10 -- data set name in JFCB not found in VTOC of this volume

11 -- data set name not unique in this volume

After changing the key (the data set name), the DSCB is rewritten by use of RETAIN.

ROUTINES USED WITH VAM FORMAT VOLUMES

FINDEXPG (CZCEL)

FINDEXPG is a reentrant, nonrecursive, privileged routine, which resides in virtual storage. This routine is entered when pages of external storage are required for a data set. (See Chart BI.)

Entry Point:
CZCEL1 -- Normal entry. Type I linkage.

Input: Register 1 contains a pointer to a parameter list:

Word 1 JFCB address

Word 2 Address of the receiving area

Word 3 Address of a word containing the number of pages required (negative if WRITDSCB is not to be called)

Output: Each page assigned will be indicated by one word in the receiving area of parameter two, in the following format.

Bits 0-15 - Relative Volume Number
Bits 16-31 - Relative Page Number

Assumptions: The format of the volume list for private data sets is exactly the same as the PVT.

Modules Called:
VOLSRCH (CZCEH) -- To find suitable volumes from which to allocate space.

RELEXPG (CZCEN) -- To return assigned pages and adjust user resource count if entire allocation cannot be made.

WRITDSCB (CZCEW) -- To update the DSCB chain.

ESA LOCK (CZCEJ) -- To lock and unlock PAT pages.

RCR (Macro) -- Used in accounting for user public resources.

READWRIT (CZCEM) -- To write PAT pages.

Exits:
Normal - Register 15 contains 00.

Error - Register 15 contains one of the following codes:

04 Insufficient space.

08 Resource limits exceeded.

The following errors cause an ABEND.

1. Error writing a PAT page

2. Illegal device code in the SDAT

Operation: The JFCB address, the address of a receiving area, and the address of the number of pages required are presented to the routine. If the data set is public, FINDEXPG passes this information to the Resource Control Routine which determines if the request is allowed, or if the user has already reached his resource limits. If RCR makes an error return, FINDEXPG returns to the calling routine with an indicator (08) in GR 15.

Otherwise, VOLSRCH is called to locate a volume from which allocation may be made. When a suitable volume has been located, the PAT is obtained, by using the PVMA in the SDAT, and locked. The PAT is scanned for invalid codes. If one is found, a message is sent either to the operator if the volume is public or to the user if the volume is private. Then VOLSRCH is called for a new volume. The PAT is next inspected for unallocated pages. As one is found, the user count is incremented in the PAT entry for this page and a word entry, consisting of relative volume number and relative page number is stored in the user's receiving area, the address of which is an input parameter. If more pages are needed, the PAT is scanned for more zero entries, as above. When either the allocation is complete, or there are no more unallocated slots in the PAT, the PAT page is written out via READWRIT and the lock released.

In the second case, the appropriate PAM bit in the SDAT is set to indicate no further space. If there are more PAT pages for this volume, they are inspected in turn. When the space request cannot be satisfied from the current volume, VOLSRCH is called at the second entry point to find another volume.

If VOLSRCH is unable to find a suitable volume, RELEXPG will be called to release any pages already assigned and the callers receiving area will be zeroed out. If the data set is public, RCR will be called to delete those pages requested but not assigned, the return code will be set to (04) and the following message will be sent to the operator:

ZCEL - NO MORE DATA PAGES AVAILABLE ON PUBLIC VOLS

The following message will then be written to the user:

ZCEL - INSUFFICIENT SPACE TO BUILD DATA SET (data set name)

When the request has been satisfied, control is returned to the calling routine with GR15 zeroed, to indicate success. Just before a successful return, WRITDSCB will be called to update the DSCB chain, unless the word containing the number of required pages is negative.

ADDDSCB (CZCEK)

ADDDSCB is a reentrant, nonrecursive, privileged routine, residing in virtual memory. ADDDSCB is called by ADDCAT, WRITDSCB, and DSCBREC to assign space for a new format E or format F DSCB. (See Chart EJ.)

Entry Points: CZCEK1 -- Normal entry.

Input: Register 1 points to a parameter list:

Word 1 Address of the JFCB associated with the VAM data set for which the DSCB assignment is to be made.

Word 2 Address of the volume table associated with this data set.

Word 3 Address of a one-word return area.

Word 4 Address of a control word that has the following format:

Bits 0-7 - X'80' if call is for a format E DSCB
X'00' if call is for a format F DSCB
Bits 8-15 - unused
Bits 16-31 - The RVN of the current DSCB page if the call is for a format F DSCB.

(Note: If the call is from DSCBREC (CZCEF) for a format E DSCB, word 4 will be set to zero.)

Output: A pointer to the newly assigned DSCB will be placed in the return area whose address is the third input parameter. The pointer is in the form:

Bits 0-3 - DSCB slot number
4-15 - Relative Volume Number
16-31 - Page number

Workpage CZCOZ will contain the new DSCB page.

Assumptions: If the call is for a format F DSCB, the device indicated by parameter four is assumed locked and will be used, then left locked upon return. The new device obtained from VOLSRCH will be locked and left locked, and the pointer of parameter four will be changed to point to this new device.

Modules Called:
VOLSRCH (CZCEH) -- To find suitable volumes from which to allocate space.

ESA LOCK (CZCEJ) -- To unlock or lock PAT pages.

READWRIT (CZCEM) -- To read and write DSCB pages and to write the PAT.

Exits:
Normal - Register 15 contains 00. Output as described.

Error - 1. No DSCB space available. ABEND.

2. Error reading or writing DSCB or PAT. Message to operator.

3. Illegal device code. ABEND.

Operation: ADDDSCB requires that, on entry, general register 1 contain a pointer to a parameter list containing:

1. The address of the JFCB associated with the VAM data set for which the DSCB assignment is to be made;

2. The address of the volume table associated with this data set;

3. The address of a one-word return area.

In addition, if the call is to request space for a format F DSCB, it is assumed that a DSCB page is being used by the calling routine and that the device containing this DSCB page is locked. In this case a fourth parameter is necessary. This fourth parameter will be the address of a word containing the RVN of the locked device. The calling routine will signify a request for a format E DSCB by making the fourth parameter zero.

To find space for a format F DSCB, ADDDSCB will scan first the DSCB page and then the volume on which the format E DSCB resides. If there are no usable pages on that volume, or if the call is for a format E DSCB, ADDDSCB will obtain a volume to search by calling VOLSRCH (CZCEH). VOLSRCH will select a suitable volume, place its relative volume number in an address supplied by ADDDSCB, set a successful return code, and return to ADDDSCB.

If VOLSRCH returns with an unsuccessful return code (no suitable volume found), ADDDSCB will call VOLSRCH again at the initial entry (CZCEH1) and this time will attempt to allocate DSCB space from any continuation slots. If, in this attempt, VOLSRCH again returns unsuccessfully, ADDDSCB will call ABEND with a message to the user.

Upon a successful return, the PAT page of the returned volume is checked for invalid codes. If one is found, a message is sent to the user if the volume is private or to the operator if the volume is public. An attempt is made to get another volume. If no invalid code is found, VOLSRCH return indicators are checked for one of two possible cases: (a) space exists on an unfilled DSCB page, or (b) all DSCB pages presently assigned are full but unassigned pages exist from which a new DSCB page may be allocated. These two cases are described below.

(Case a) SDAT PSM indicates unfilled DSCB page(s) exist:

VOLSRCH returns a volume pointer and indicates the PAT page for that volume which describes an unfilled DSCB page. A search of this PAT page is made for an unfilled DSCB page entry. If no unfilled DSCB pages exist, it indicates that the PAT Summary Table (PST) must be updated and a new DSCB page assigned from this PAT if possible. See case (b) for this procedure. When an apparently usable DSCB page entry is located, the DSCB page on external storage is read into the workpage CZCOZ by a call to CZCEM. If this DSCB page was just assigned (see case (b)), the PAT entry

has already been updated; the first DSCB slot in the page is assigned by setting bits 6-7 of DSETYP of the DSCB slot to 01 for format E or 10 for format F. The DSCB is then checksummed and written back to external storage via CZCEM. The new DSCB assignment is passed back to the calling module in the form of relative volume number, page number and DSCB slot number.

If this DSCB page was not a newly assigned page, the DSCB page is searched for an available DSCB slot (bits 6-7 of DSETYP equal to 00). If assigning a slot will leave four or more available slots on the page, the slot is assigned. If it leaves just four, the PAT entry will be changed from '80' to '82'. A page so marked will not thereafter be searched except to find space for a format F DSCB whose format E DSCB is on that page, or if no other space is available. If no available slots are found in this DSCB page, the PAT entry is set to '83' to indicate a completely full page and the search for another page continues. When an available slot is found, it is assigned by setting bits 6-7 of DSETYP. The DSCB is checksummed and, if call is for a format E DSCB, written back to external storage via CZCEM. If the PAT page has been changed, it is written back to external storage via CZCEM. The new DSCB assignment is passed back to the calling module as noted in the previous paragraph. If the calling program is ADDCAT (CZCFA), any devices locked by ADDDSCB will be unlocked before returning.

(Case b) SDAT PST indicates all DSCB pages full but unassigned pages exist.

When VOLSRCH returns a volume pointer and indicates a PAT page from whch a new DSCB page may be assigned, ADDDSCB searches that PAT page and, upon finding an unassigned page, assigns that page as a DSCB page by setting the PAT entry to '80'. The SDAT PST DAM (DSCB Availability Mask) is updated to reflect this newly assigned DSCB page. The new page is read into workpage CZCOZ. All DSCB slots on the page are made available for future assignment by setting them to zero. DSCB assignment then continues as described in case (a) for a newly assigned DSCB page.

If no unassigned pages are available for DSCB assignment, the PST is updated to reflect the fact that no more space is available on the volume page described by the PAT page under examination. ADDDSCB will then examine the next PAT page (if more than one exist) for the volume and repeat the above process. If no space can be found on this volume, VOLSRCH is reentered at its second entry point (CZCEH2), and a new volume is obtained. The entire procedure is then repeated.

VOLSRCH (CZCEH)

VOLSRCH is a privileged, reentrant nonrecursive routine which is called by ADDDSCB and FINDEXPG to determine the most suitable volume in a given list, public or private, from which to allocate space. (See Chart BK.)

Entry Points:
CZCEH1 -- For initial call.
CZCEH2 -- For subsequent calls.

Input: Register 1 contains a pointer to a parameter list:

Word 1 Address of control word. (Control word format is as follows:

Byte 0 - X'00' DAM, PAM search
- X'80' PAM search only
- X'C0' Primary allocation set by OPENVAM (CZCOA)

Bytes 1-3 - Number of pages requested by FINDEXPG (CZCEL); Not used when Byte 0 is X'00'.)

Word 2 JFCB adress.

Word 3 Address of the halfword return slot for the relative volume number.

Word 4 Address of the volume list.

Output: The relative volume number of the selected volume will be placed in the halfword which is addressed by word 3 of the input parameter list. Register 1 contains a pointer to the input parameter list.

Restrictions: The second entry point (CZCEH2) may be entered only after the first (CZCEH1) has been entered.

The task will SYSER and ABEND if the data set is public and there are no volumes in the PVT.

Modules Called:
SYSER (CEAIS) -- Full VM dump.

ABEND (CZACP) -- Terminate task and return control to the terminal.

Interlock (CZCOH) -- To lock RESTBL.

Release Interlock (CZCOI) -- To unlock RESTBL.

Exits:
Normal - Register 15 contains 00.

Error - Register 15 contains one of the following codes:

04 No space on any volume.
08 CZCEH2 called out of turn.

Operation: The first input parameter is the address of a control byte which indicates the type of search required. If the control byte contains an '80', VOLSRCH will examine the PAM bits of each volume to find one containing available data pages. If the control byte is '00', VOLSRCH will test each volume, examining first the DAM bits to find one containing an available DSCB page. If the DAM bits indicate that no such page is available, the PAM is then examined to see if a data page is present which may be assigned as a DSCB page. If one is, the control byte is changed to '80' to inform the calling routine that such is the case. If there are none, the next volume is then examined. If the second bit of the control byte is on, VOLSRCH is being called for primary allocation and the RESTBLE will not be searched to determine on which volumes the data set is currently located.

Whenever a suitable volume is found, the relative volume number is placed in the halfword return slot whose address is the third input parameter. If no suitable volume is found, return will be made to the calling routine with general register 15 set to '04'.

For a private new data set, the volume list is examined for the device having the most available space. For an existing private data set, the search begins with the volume containing the format E DSCB, and cycles from the last in the list to the first.

For public data sets, the volume entries in the JFCB will be examined for valid, mounted, public volumes. If any are found, the search will be restricted to them. If none are found in the JFCB, the public volume table will be searched.

For public data sets, the first volume of the PVT is reserved for system use and will be examined last. For a new public data set, a search will be made for the device having the most available space. For existing public data sets, the search begins with the volume containing the format E DSCB.

When the data set is duplexed, the volumes containing the duplexed copy are not examined unless there is no space on any other public volume, including the system volume.

If the calling routine needs more than one volume, the second entry (CZCEH2) should be called. VOLSRCH will then continue the search. If multivolume data sets have been inhibited, data sets will not be allowed to expand to more than one device. If an old data set is already multivolume, it will be restricted to those volumes on which it already exists.

RELEXPG (CZCEN)

RELEXPG is a reentrant, nonrecursive, privileged routine residing in virtual storage, which is called by CLOSEVAM, DELVAM, WRITDSCB, DSCBREC, and FINDEXPG when external pages and DSCBs are to be made available. (See Chart BL.)

Entry Point: CZCEN1 -- Normal entry by type I linkage.

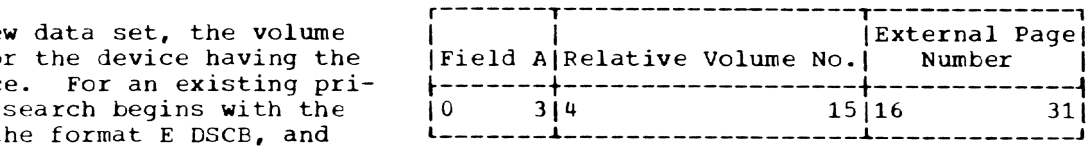
Input: Register 1 points to the following parameter list:

Word 1 JFCB address

Word 2 Address of the list of entries to be released.

Word 3 Address of a word containing the number of entries in the list.

The format of entries in the release list is:



The release list must be aligned on a fullword boundary. Field A contains a DSCB slot number when a DSCB is being released.

Output: Relevant PAT entries and DSCB pages will be changed to reflect the requested releases.

Assumption: Private data sets will have a volume list in exactly the same format as the PVT.

Modules Called:
CZCOY -- Workpage for reading DSCB pages.

RCR -- To credit user with pages being returned.

ESA LOCK (CZCEJ) -- To lock and unlock devices.

READWRIT (CZCEM) -- To read and write DSCB pages and to write PAT pages.

Exits:
Normal - Register 15 contains 00.

Error - None.

Operation: The routine is entered with the address of the JFCB, the address of the list of entries to be released (these can be mixed data pages and DSCBs), and the address of a word containing the number of entries in the release list. These last two parameters are set up as indices and a loop count and the first entry in the release list is obtained.

The relative volume number is stored in a comparison field so that the current PAT page can be used to release consecutive entries for the same relative volume before being rewritten. The relative volume number is used to index the volume list; the SDA is extracted and used to calculate the appropriate SDAT entry.

The external page number of the entry in the release list is examined for validity by type of device and set to modulo 4096 to get the relative PAT page and entry within the page. The device is locked and the virtual memory address of the PAT obtained using the PVMA field in the SDAT.

The PAT is scanned for invalid codes. If one is found, a message is sent either to the user if the volume is private or to the operator if the volume is public. If the PAT is invalid, it will not be changed or written.

The PAT entry is examined. If it is 80, 82, or 83 (a DSCB page) the proper DSCB page, if not already in core, will be read in. The indicated slot will be released and the PAT entry changed, if necessary, according to the number of slots still assigned. If no slots are left assigned, the PAT entry will be set to zero, making the entire page available, and the appropriate PAM bit set to show at least one available page entry. Otherwise, the relevant DAM bit will be set to show at least one DSCB available.

When a data page is being released, the count of users is decremented. If the resulting count is non-zero, the page cannot be released. Otherwise, the PAM bit is set to show at least one available page entry in this PAT page.

If the PAT entry of the page to be released is 'C0', an error entry, the relocation field will be searched to find the relocated page number. If none is found, SYSER 002 is called and the program continues with the next entry to be released.

A special feature of RELEXPG allows external PAT page entries to be marked as error pages. When a data page is being released, bit 1 of the release entry is examined. If it is on, the PAT page entry is set to C0, thus making it an error entry.

As pages are made available, PVTAVS, the number of available pages is updated.

Processing continues by incrementing indices and decrementing the count of entries in the release list. When more entries are to be released, the relative volume number of the next entry is compared with that of the previous entry. If these are equal, the routine will loop to process this entry. Otherwise, any altered PAT pages must be rewritten using READWRIT and this relative volume number saved. The routine then loops to use the relative volume number as an index to the volume list.

When all entries have been processed, the altered PAT pages on the current volume are written using READWRIT. The Resource Control Routine is called to adjust the resource limits when any public pages are returned. RELEXPG then returns to the calling routine.

DSCBREC (CZCEF)

DSCBREC is a reentrant, nonrecursive, privileged routine residing in virtual memory. This routine is used to recover as far as possible from a checksum error. (See Chart BM.)

Entry Points: CZCEF1 - Normal entry via type I linkage.

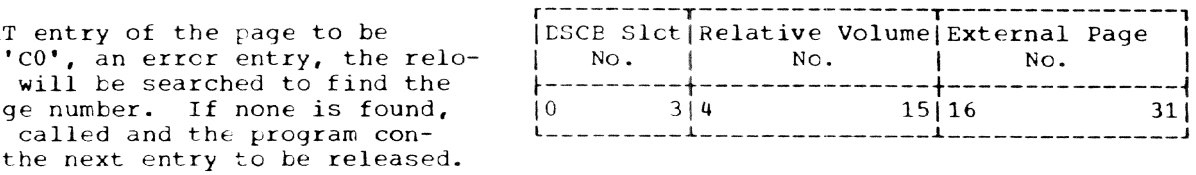
Input: Register 1 contains a pointer to the follcwing parameter list:

Word 1 JFCB address

Word 2 Address of a fullword describing the bad DSCB, which will be changed upon successful return to the address of a replacement DSCB which should be re-read.

Word 3 Address of the Volume Table.

The format of the second parameter word is:



Output: Register 1 contains a pointer to the input parameter list.

Modules Called:
ADDCAT (CZCFA) -- To update DSCB pointer in the DSD.

ADDDSCB (CZCEK) -- To supply new DSCB slots.

DELCAT (CZCFD) -- To delete a catalog entry.

DSCB/PAT RECOVERY (CZUFX) -- To rebuild a user catalog from public DSCEs.

FINDEXPG (CZCEL) -- To obtain new data pages.

RELEXPG (CZCEN) -- To release to the system DSCB slots and data pages.

ESA LOCK (CZCEJ) -- To lock and unlock devices.

RCR -- Resource control macro to credit the data set owner with lost public pages.

GETMAIN/FREEMAIN (CZCGA) -- Get or free virtual storage.

SETXP (SVC 244) -- To read a data page.

PGOUT (SVC 242) -- To write a data page.

READWRIT (CZCEM) -- To read and write DSCB pages and write the PAT.

USERCAT SCAN (CZUFY) -- To compute the checksum for the SYSSVCT DSCB and to construct the first page of the SYSSVCT data set if necessary.

Exits:
Normal - Register 15 contains 00.

Error - Register 15 contains one of the following codes:

04	Recovery is impractical - (not duplexed or, data set shared and format E DSCB bad).
08	Not enough pages available.
0C	Unrecoverable write error encountered.
10	Error page found with no relocation entry.

Operation: DSCBREC requires, on entry, a pointer in General Register 1 to a parameter list containing the address of the JFCB with which the DSCB in error is associated, the address of a pointer to the DSCB, and the address of the volume table.

Upon entry, the following message will be written to the operator:

ZCEF - CHECKSUM ERROR ON SLOT XX, PAGE XXXX, R.V.N.XX, V.S.N.XXXXXX

Following this, DSCBREC will check to see if the DSCB in error is the format E DSCB of a shared data set. If so, complete recovery is not practical. If not, a check will be made to see if the calling program is WRITDSCB. WRITDSCB calls DSCBREC only if the format E DSCB is found to be bad. No attempt to "save" the data set is required as all data page pointers are present in the RESTBL and the DSCB chain is about to be updated anyway. In this special case DSCBREC will assign a new format E DSCB, update the E DSCB pointers in the RESTBL, JFCB and DSD, then checksum and write to external storage the new DSCB slot. If the calling program is not WRITDSCB, a check will be made as to whether or not the data set is duplexed.

If the data set is not duplexed, complete recovery is not practical but an attempt will be made to save as many pages as possible. DELCAT will be called to delete the data set from the catalog and the user will be notified that this has been done. The DSCB chain will then be scanned and all DSCBs preceding the bad one, together with their described data pages, will be returned to the system via RELEXPG. All other DSCBs and data pages in the chain will be lost. The number of lost pages will be calculated and credited to the user via the Resource Control macro. The user or the operator, depending upon whether the data set is on private or public volumes, will be informed of the number of pages lost. Return is then made with '04' in register 15.

If the data set is duplexed, the DSCB chain will be searched to find the relative position of the bad DSCB and the number of pages that will be lost. The Resource Control macro will be called to credit the user with the number of pages lost and FINDEXPG will be called to replace these pages. At this time TDTDCI will be checked to see if this is the primary or the secondary copy.

If it is primary, a dummy secondary RESTBL will be constructed from the secondary DSCB chain and the data set will be transferred from these addresses to those supplied by FINDEXPG for the primary data set.

If it is the secondary copy, the same procedure will be followed except that no dummy RESTBL need be constructed as the primary RESTBL already exists. The data set will be transferred from addresses in the primary RESTBL to those supplied by FINDEXPG.

Once this transfer is complete, DSCBREC will check the format type of the bad DSCB. If it was a format F DSCB, the DSCB string will be updated from those pages found by FINDEXPG. If it was a format E DSCB, DSCBREC will first update the DSCB header

and the E DSCB pointers in the JFCB and DSD.

If the data set is shared, DSCBREC has no knowledge of the location of the sharing JFCBs and is therefore unable to update their format E DSCB pointers. Therefore, if the bad DSCB is format E and the data set is found to be shared, DSCEREC will consider the data set as one that has no duplex copy. The data set will be deleted via DELCAT and register 15 will hold the return code of '04'.

Before exiting, the following messages will be printed:

1. To the operator (if data set is public):

   ZCEF - RECOVERY (SUCCESSFUL.XXXX)/ (IMPOSSIBLE.ALL) PAGES LOST

2. To the user:

   ZCEF - CHECKSUM ERROR ON SLOT XX, PAGE XXXX, R.V.N. XX, V.S.N. XXXXXX

   ZCEF - RECOVERY (SUCCESSFUL.XXXX)/ (IMPOSSIBLE.ALL) PAGES LOST

3. To the user if recovery was not successful:

   ZCEF - CHECKSUM ERR HAS CAUSED DELETION OF DATA SET (d.s.name)

WRITDSCB (CZCEW)

WRITDSCB is a reentrant, nonrecursive, privileged routine which resides in virtual memory. WRITDSCB is called by OPENVAM, CLOSEVAM, and FINDEXPG to construct a DSCB chain from the RESTBL, JFCB, DCB, and PVT. (See Chart BN.)

Entry Points: CZCEW1 - Normal entry. Type I linkage.

Input: General Register 1 contains a pointer to the parameter list:

Word 1 Address of the RESTBL reader

Word 2 Address of a control byte

bits 0-3 - 'F' - new data set
'0' - old data set

bits 4-7 - 'F' - write all pages in RESTBL
'0' - write only pages in use

Word 3 Address of the JFCB

Word 4 Address of the DCB

Output: Register 1 contains a pointer to the input parameter list.

Assumptions: The format of the volume lists of both public and private data sets are identical.

Modules Called:
ADDDSCB (CZCEK1) -- To find a new format E or F DSCB.

RELEXPG (CZCEN1) -- To return excess DSCB slots.

DSCBREC (CZCEF1) -- To set up new format E DSCB.

ESA LOCK (CZCEJ) -- To lock and unlock devices.

READWRIT (CZCEM) -- To read and write DSCB pages and to write the PAT.

Exits:
Normal - Register 15 contains 00.

Error - 1. Error reading DSCB page. ABEND.
2. Error writing DSCB page or PAT. ABEND.
3. Unrecoverable checksum error. ABEND.

Operation: Input parameters for WRITDSCB are the address of the RESTBL and the address of a control byte which is "FF" for a new data set and '00' for an old one. If the data set is new, two additional parameters, the address of the JFCB and the DCB are also required.

After initial housekeeping activities, the page containing the format E DSCB will be read in using subroutine 1. If a checksum error is encountered and the data set is not being shared, DSCBREC will be called to provide a new format E DSCB. If the return from DSCBREC indicates an unsuccessful recovery or if the data set is shared, WRITDSCB will and ABEND with appropriate messages.

Once a good format E DSCB has been read in, the page totals in the DSCB header will be updated from the RESTBL header. At this time also, if the data set is not partitioned the DSCB header will be filled in from information in the JFCB and DCB.

If the data set resides on private volumes, the number of volumes in the DSCB will be checked against the number of private volumes. If they are equal or if the volume count is equal to one, no volume updating is required. If they are not equal, the DSCB volume count will be updated from the number in the PVT. Volume

IDs from the PVT will then be moved into the format E DSCB. If more DSCBs are required, the next DSCB in the chain will be used. If there is no entry in the chain field, subroutine 3 will be called to get a new one. This will continue until all volume IDs have been entered.

Following any volume updating, WRITDSCB will begin updating page entries in the DSCB from the RESTBL, obtaining any additionally needed DSCBs as in the volume updating procedure. Whenever a DSCB has been filled and its chain field updated, its checksum will be computed and stored using subroutine 2. When all extents have been moved into the DSCB chain, any unfilled entry space in the last DSCB will be set to zero and any unused DSCBs will be released via RELEXPG.

Subroutine 1: This subroutine will read in the page containing the indicated DSCB and set the base register for that DSCB. The checksum will then be tested and a branch will be made accordingly to either the normal or the error return.

Subroutine 2: This subroutine has two entries. When the first entry is used, DSETYP will be checked to see if this is a format E DSCB. If it is not, DSETYP will be set to indicate a format F DSCB. The checksum value for the DSCB will then be computed and placed in the checksum field (DSECKS). When the second entry is used, only the checksumming will be done.

Subroutine 3: This subroutine will search the currently held DSCB page in CZCOY for a usable DSCB slot. If one is found, it will be assigned and the PAT entry updated as necessary. If one is not found, the PAT entry will be set to indicate a full page and ADDDSCB will be called to furnish a new DSCB slot. The value returned from ADDDSCB will be placed in the chain field (DSECHN) of the previous DSCB, that DSCB will be checksummed via Subroutine 2, and the old DSCB page will be written out. The new DSCB page, placed in CZCOZ by ADDDSCB, will be transferred to CZCOY.

VAMINIT (CZCEQ)

VAMINIT is a privileged, reentrant, nonrecursive routine residing in virtual memory. This routine initializes private VAM volumes when they are entered into the system. (See Chart EO.)

Entry Point: CZCEQ1 via type I linkage.

Input: Register 1 contains the address of the SDAT entry for the volume to be initialized.

Output: The tables associated with the volume are updated.

Restriction: VAMINIT must not be invoked for a public volume.

Modules Called:
SETXP (CEAH7) -- To read the PAT into virtual storage.

GETMAIN (CZCGA2) -- To get the virtual storage for the PAT.

OBTAIN (CZCFO1) -- To read the volume label.

ABEND (CZACP) -- Returns control to the terminal.

Exits:
Normal - Register 15 contains 00.

Error - ABEND -- Routine was invoked for a public volume.
ABEND -- Device code other than 2311 or 2314.

Operation: VAMINIT is entered with Type I linkage and is passed the following parameter:

Address of SDAT entry

After a normal initialization procedure (storing registers, setting up base registers, etc.), VAMINIT will check SDAT location SDAPP to determine whether this is a private or public volume. If public, VAMINIT will invoke the ABEND procedure with an appropriate error message.

If private, VAMINIT will have the volume label read into virtual memory by invoking the OBTAIN routine. From information in the volume label VAMINIT will then enter the volume ID and the PAT origin (PTO) in the SDAT.

VAMINIT will then invoke the GETMAIN routine to obtain virtual memory for the page assignment table (PAT). If the device is a 2311, PAT will be one page in length; if 2314, two pages in length. The PAT summary mask (PSM) in the SDAT is set up and VAMINIT invokes the SETXP routine to read in the PAT. After reading in the PAT, VAMINIT will set the VAM/SAM flag (SDAAM) to 0 and set up the PAT Virtual Memory Address (PVMA).

VANINIT will verify that the PAT pages read in by SETXP are valid. If they are not valid, a message will be written to the system log and to the user, and the invalid PAT bit will be set in the SDA.

VAMINIT will then return to the routine which called it via Type I return linkage.

READWRIT (CZCEM)

READWRIT is a reentrant, nonrecursive, privileged routine residing in virtual storage. It is used to read DSCB pages into virtual storage or write DSCB or PAT pages to external storage. (See Chart BP.)

Entry Point: CZCEM1 - Normal entry via Type I linkage.

Input: Register 1 contains the address of the following parameter list:

Word 1 Address of the volume table or, if none exists, of the halfword containing the SDA number.

Word 2 Address of a one-byte request flag. The first half-byte is '0' to write a DSCB page, '4' to read a DSCB page or '8' to write the PAT. The second half-byte is in the following format:

bit 1 unused.

bit 2 = 0, parameter 1 has address of PVT.
= 1, parameter 1 has address of SDA no.

bit 3 = 0, slot no. in parameter 3 used.
= 1, slot no. ignored.

bit 4 = 0, address of JFCB in parameter 5.
= 1, parameter 5 contains address of FQN or is ignored.

WORD 3 Address of a one word pointer to the DSCB or RVN in the following format:

bits 0-3 DSCB slot number.

bits 4-15 Relative volume number.

bits 16-31 Relative page number.

For a write DSCB request, the slot number will always be ignored. For a write PAT request, only the relative volume field will be used.

Word 4 Address of a page boundary buffer into or from which the DSCB page will be read or written. Parameters 4 and 5 are not required for a write PAT request.

Word 5 Address of the JFCB or, if there is none, the address of the fully qualified data set name (FQN).

Output: If the requested DSCB page is found to have been relocated, the new page number will be used to update the DSCB pointer supplied in parameter 3.

Assumptions: It is assumed that the device to be read from or written to has been locked, if necessary, prior to the call to READWRIT. No device locks will be set or reset within READWRIT. However, if the request is to write to a public device, a check will be made to be sure the device has been locked by this task.

Modules Called:
SETXP (SVC 244) -- To read DSCB pages.

PGOUT (SVC 242) -- To write DSCB and PAT pages.

Exits:
Normal - Register 15 contains 00.

Error - Register 15 may be set to one of the following return codes:

04 Indicated volume beyond PVT limit.
08 JFCB and RESTBL DSCB pointers disagree.
0C Relocation entry cannot be found.
10 Pointer indicates a non-DSCB page.
14 Checksum error on indicated DSCB slot.
18 Data set names disagree.
1C Unable to write PAT.
20 Unable to write DSCB page.
24 Device not locked by this task for a write operation.
28 Page number beyond device limits.
2C Invalid buffer area address.

Operation: If the PVT address is given, the input RVN is checked to see that it is within PVT limits and the SDA number obtained from the proper PVT entry. The SDA number is then used to compute the SDAT entry address. If the request is to write the PAT, control passes to the write PAT subroutine.

Otherwise, the address of the buffer area (parameter 4) is checked to be sure it is non-zero and on a page boundary. The page number to be read or written is checked against the limits for that device. If the JFCB has been supplied (in parameter 5), and if a RESTBL exists for this data set, the DSCB pointers in the JFCB and RESTBL are compared to be sure they agree. The PAT entry is then checked to see that it indicates a DSCB page. If the entry is 'C0', READWRIT scans the PAT relocation entries to find the correct page. If the request is to write the DSCB page, control

is passed to the write subroutine. If not, the DSCB page is read in. If the slot number has been specified, the checksum is computed and compared and, if this is a completed format 'E' DSCB, the data set name is compared to that in the JFCB or the FQN supplied in parameter 5.

READWRIT then returns control to the routine which called it via type I return linkage.

Write DSCB Subroutine: If the device to be written to is public, the SDAT entry is checked to see that it has been locked by this task. The page is then written out via PGOUT.

Write PAT Subroutine: If the device is public the SDAT entry is checked to be sure the device was locked by this task. The PAT page number is obtained from the SDAT and compared with the page limits for this device. The PGOUT parameter list is then filled in from the SDAT and the PAT pages written.

Any error found results in a diagnostic message to the operator and a nonzero return code to the caller. (See Exits.)

ESA LOCK (CZCEJ)

ESA LOCK is a reentrant, nonrecursive, privileged routine residing in virtual storage. It is called to set and clear virtual memory locks and to record the task and module which applied the lock. CZCEJ is called for the SDAPLO lock by all routines reading and writing DSCB and PAT pages. (See Chart BQ.)

Entry Points:
CZCEJ1 -- to set lock.

CZCEJ2 -- to clear lock.

CZCEJ3 -- ABEND reset of locks.

Input: Register 1 points to the following parameter list:

Word 1 Address of a parameter word formatted as follows:

byte 1 - Type code. (X'01' for the SDAPLO lock.)

byte 2 - Wait count. This specifies the number of TSEND's to wait for the lock to be released. When the lock cannot be set in the requested time, control is returned to the caller with return code set to X'04'.

byte 3 - Lock code. If this byte is set to X'80', the calling task is allowed to lock more than two PATs.

byte 4 - Reserved

Word 2 Address of the SDAT entry.

Output: Lock processing will be performed as requested.

Modules Called:
PAIR (CZACS) -- To put an entry in the AIR table.

SYSER (CEAIS) -- Minor system error.

Exits:
Normal - Return to calling routine with register 15 set to 00.

Error - Return to calling routine with return code 04; lock could not be set in limited attempts requested.

Operation: The type code is first edited to determine the type of lock to be set. Currently any code but X'01' (SDAPLO lock) will cause a SYSER. PAIR is then called to put the address of the ABEND reset routine (CZCEJ3) into the AIR table. Entry is then made to the appropriate lock set or lock reset routine.

The lock set routine first determines whether the lock is already set. If set by this task it returns control to the calling routine with return code of 00. If set by another task it forces time slice ends until the lock is cleared or until the maximum number of TSEND's specified by the wait count parameter is reached. In the latter case, it exits to the calling routine with return code 04.

If the lock is clear, the lock set routine determines whether this task is allowed to lock more than two PATs. If so, the lock is set; the task ID and PSECT address of the routine setting the lock are recorded in the SDAT and control is returned to the calling routine with return code 00. If the task is not allowed to set more than two PATs, the lock set routine determines whether the task already has the maximum number of locks allowed (currently two for SDAPLO). If so, SYSER is called. If not, the lock is set; the task ID and PSECT address of the routine setting the lock are recorded in the SDAT and control is returned to the calling routine with return code 00.

The lock reset routine first determines whether the lock was set by this task. If not, SYSER is called. It then checks if it was set by this module. If not, it returns

control to the caller with return code 00, but does not reset the lock. If locked by this module, the lock reset routine clears the lock task ID and PSECT address and returns control to the caller with return code 00.

The ABEND reset routine is called by ABEND to clear all locks set by the task if an abnormal terminate becomes necessary.

Device management allocates, mounts, and releases private devices used by all tasks except the BULKIO task. Devices required by the BULKIO task are allocated by BULKIO; MTREQ is called only to update tables in this case. At system startup, each device attached to the time sharing system is designated as either public or private. Public (system) devices are assumed to be permanently mounted. Device management also restricts tasks to the limits allowed for private devices, and maintains charges for their use.

Five routines make up device management as follows:

MTREQ - allocates devices, restricts tasks to preestablished resource limits, and starts charges for private devices.

PAUSE - issues mount requests to the operator and awaits his response and validates the suitability of the mounted volume.

RELEAS - releases a device that has been allocated, services requests for devices enqueued on the request queue, and updates and/or stops charges.

BUMP - called to mount subsequent volumes of a multi-volume SAM data set.

MOUNTVOL - initializes, builds the PVT for, and calls MTREQ to mount, all volumes of a VAM private data set.

GENERAL OPERATION

For a conversational task, the DDEF command routine will call MTREQ each time a private device is requested for non-VAM data sets. If a private device is requested for a VAM data set, the DDEF command routine will invoke MOUNTVOL to initialize the PVT. MOUNTVOL will call MTREQ and return control to the DDEF command routine. If the USER's ration allows the allocation, MTREQ first checks to see if a device of the required type is available and whether or not the desired volume is already mounted. If the device and volume are ready, allocation is carried out at once. If a device is available but the required volume is not mounted, MTREQ calls the PAUSE routine to issue a mount request to the operator. Allocation is completed as soon as the operator indicates that mounting was done. Finally, if the USER's ration allows, and a device of the required type is not available, MTREQ places an entry in the request queue table. This table is scanned by the RELEAS routine each time it releases a device, to see if the just-released device has been requested. If requested, it allocates the device and passes to MTREQ for table updates.

Note that every task needing a private device has its own copy of the MTREQ routine. If that routine finds a requested private device unavailable, it adds a request to the request queue table, which is shared by all tasks, and then puts itself and its task into wait status. When the RELEAS routine operates, it scans the request queue and interrupts the first task awaiting the just-released device according to a three level priority; SDA requests first, conversational requests second, and all other requests in the third level.

Device management operations for a nonconversational task are similar except that every private device required by the task must be allocated before the task is allowed to proceed. To do this, device management performs in two phases. In the first phase, the SECURE command routine (responding to a SECURE command) calls MTREQ to allocate every private device needed by the task. Only when the allocation is complete does the nonconversational task start execution; until then, the task is suspended. In the second phase, when the nonconversational task is executing, the DDEF command routine calls MTREQ or MOUNTVOL for each private device request, asking that the desired device and volume be allocated from the set of devices secured for the task. On each call, MTREQ makes an allocation from the set of private devices it has secured for that task. This continues until the nonconversational task terminates. During the second phase MTREQ will determine if the volume is mounted elsewhere, and if so, will exchange the reservation for the device on which the volume is mounted.

MTREQ handles the mounting of VAM volumes or the mounting of an initial SAM volume, as well as the allocation of unit-record devices. Mounting of subsequent volumes of a multi-volume SAM data set is the function of the BUMP routine. BUMP calls PAUSE to ask the operator to dismount the currently mounted volume and replace it with the next volume. Only one device is assigned to a SAM data set, since only one volume is mounted at a time.

The general flow of device management is illustrated in Figure 5.

MOUNTVOL Routine (CZCAM)

MOUNTVOL is a reentrant, nonrecursive, privileged routine which resides in virtual storage. This routine will mount and initialize all volumes of a VAM private data set and build for it a Private Volume Table to be in the same format as a Public Volume Table. (See Chart CE.)

Entry Point: CZCAM1 via type I linkage.

Input: Register 1 contains a pointer to the parameter list:

Word 1 Pointer to JFCB

Word 2 Pointer to DSD (optional)

Word 3 Pointer to a fullword return area

Output: Register 1 contains a pointer to the input parameter list. A pointer to the PVT is placed in the third input parameter word.

Restrictions: This routine can be used only on PAT formatted private volumes.

Modules Called:
MTREQ (CZCAA) -- To mount all volumes in the JFCB.

VAMINIT (CZCEQ) -- To initialize the SDAT entry of a mounted VAM volume.

LOCATE (CZCFL) -- To retrieve the DSD of a data set if one was not provided as an input parameter.

SETXP (SVC 244) -- To prepare to read an external page into virtual storage.

GETMAIN (CZCGA) -- To obtain space for the PVT and the JFCB volume field extensions.

SYSER (SVC 228) -- To provide a full virtual storage dump.

SECURE -- To allocate the devices needed for the task, if BULKIO.

READWRIT (CZCEM) -- To read DSCB pages and to write PAT pages.

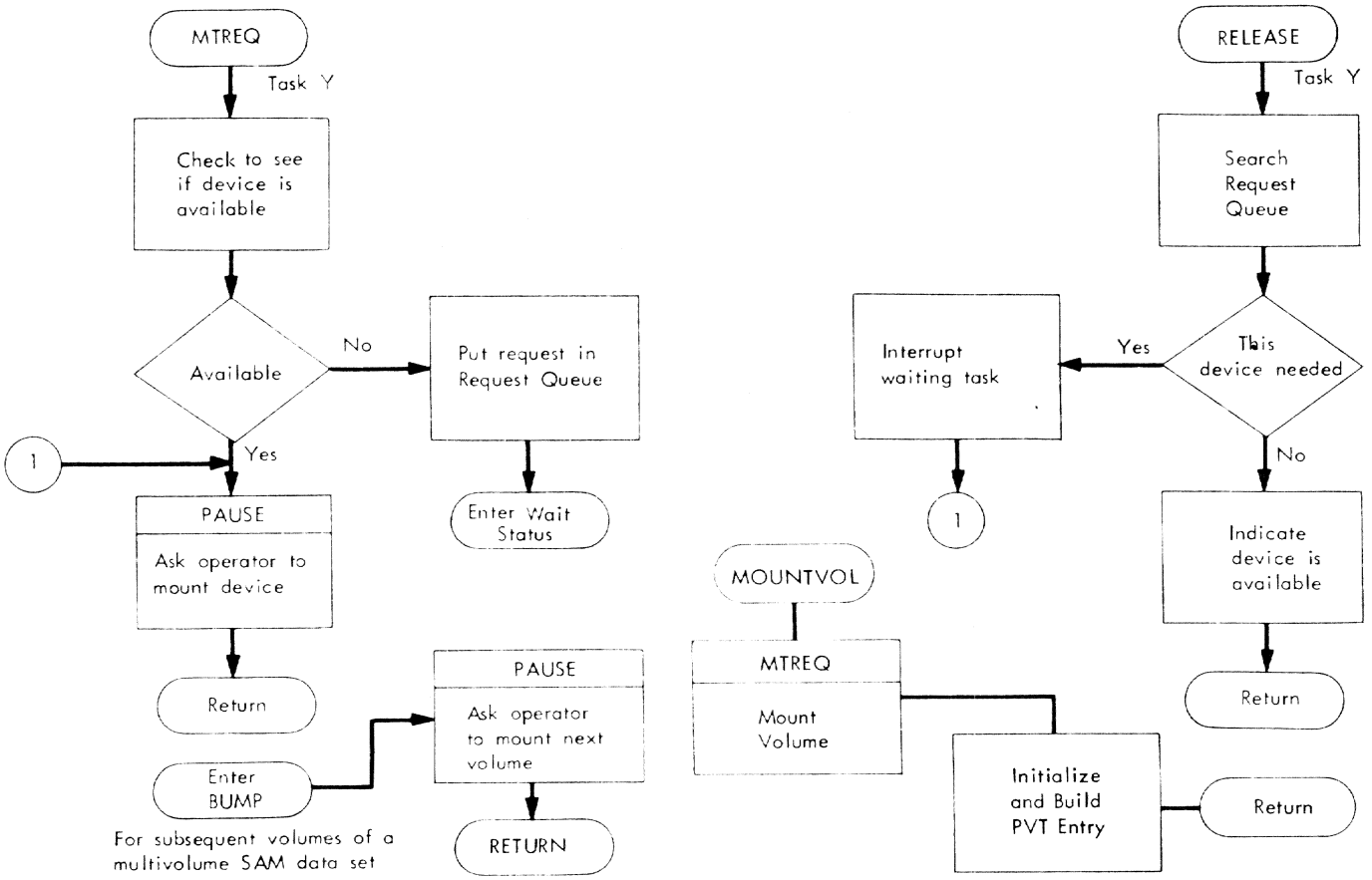


Figure 5. General Diagram of Device Management Operation

Exits:
Normal - Register 15 contains 00.

Error - Register 15 contains one of the following return codes:

04	Not a VAM data set.
08	Not a private data set.
0C	DSD not found by LOCATE, or unmounted volume returned from MTREQ.
10	Non-zero return code returned from MTREQ or LOCATE.
14	More than three volumes in JFCB appendage.

Operation: Upon entry the JFCB field, TDTDSV, is checked for data set organization. If other than VAM organization, a return to the caller is made with a code of '04'. MOUNTVOL assumes it will only be called for private data sets.

If a RESTBL is indicated by the JFCB and a PVT is indicated by RESTBL field RHDVTA, the PVT pointer is placed in the return parameter list and a return is made to the caller with a code of '00'.

The disposition field, TDTDSP, is examined. If zero, the data set is 'new'; if non-zero, the data set is 'old'. In either case, if the task is BULKIO, SECURE is called to allocate the devices needed for the task.

For new data sets, MTREQ is called to mount all volumes indicated by the JFCB volume fields. If a non-zero return code is received from MTREQ, a call to RELEAS is issued if one or more of the volumes in the JFCB is mounted. Control is returned to the calling routine with a return code of '10'. On a zero return from MTREQ, PVT size is calculated and GETMAIN is called to acquire the PVT space. The volume count is placed in the PVT header. A loop is set up to initialize each volume entry, build the PVT entries, and set the flag field if the PAT table indicates that relocated PAT pages exist. All assigned pages which are not in use are set to available in the PAT. VAMINIT is called to initialize each volume. After the volume is initialized, the SDAT entry is used to build the PVT entry (VSN, dev code, and SDA). The ordering of the volume entries in the JFCB is the ordering of the entries in the PVT. While scanning the JFCB volume fields if an unmounted volume or different number of volumes than the JFCB volume count are found, a SYSER (minor software) occurs and a call to RELEAS is issued for all the mounted volumes followed by a return to the calling routine with a return code of '10'. After the PVT is built, a pointer to it is placed in the return parameter list and a return is made to the caller with a code of '00'.

For an old data set, MOUNTVOL will first check the JFCB to determine whether the user specified any volumes in his DDEF. If he did, the volume IDs are moved into a save area to be examined later for duplication or addition.

The parameter list DSD pointer is examined; if zero, LOCATE is called to return the DSD of the data set. If a non-zero return code or other than a DSD is returned by LOCATE, SYSER (minor software) is invoked and a return is made to the caller with a code of '0C'.

Once the DSD is located, the JFCB volume count is set initially to one, and the first JFCB volume field is set up with the DSD volume ID. Task Common is checked to see if the BULKIO flag is set; if it is, MTREQ must be called with an SDA request, using the SDA which will be in the JFCB, to mount the first volume of the data set. If the BULKIO flag is not set, MTREQ will be called with the usual JFCB request to mount that volume. If a non-zero code is returned, a call to RELEAS is issued if one or more of the volumes in the JFCB is mounted. Control is returned to the calling routine with a return code of '10'. If the volume was already mounted and SDAPVMA has been initialized, the call to VAMINIT will be bypassed; if not, VAMINIT is called to initialize the first volume. A dummy one-entry PVT is built using this volume and the PVTFLG is set to '80' if the PAT table indicates any relocated pages. The READDSCB subroutine is called to read the 'E' DSCB pointer from the DSD using the VAM workpage (CZCOY) for the DSCB page.

The volume count in the 'E' DSCB is examined. If the count is zero, the data set resides on only one volume which is already mounted. If no additional volumes were specified in the JFCB, GETMAIN is called to get 32 bytes for a one-entry PVT. The PVT is built from the previously built dummy PVT, the pointer is placed in the return parameter list, and a return is made to the caller with a code of '00'.

If the DSCB volume count is greater than one, the DSCB string contains a volume entry for each volume of the data set - including the first volume. The volume count plus one, times sixteen, is used to calculate the space required for the PVT. GETMAIN is called to acquire the PVT space. The volume entries of the first DSCB are transferred to the JFCB volume fields. GETMAIN is called to acquire all the space needed for the JFCB volume field appendages (number of volumes divided by 3 and multiplied by 32). Then, as each 32-byte ap-

pendage is needed, it is taken from this area. When all volume entries of the DSCB are transferred to the JFCB volume fields, MTREQ is called with a request to mount all volumes just entered. If a non-zero return code is returned by MTREQ, a call to RELEAS is issued if one or more of the volumes in the JFCB is mounted. Control is returned to the calling routine with a return code of '10'. Otherwise a loop is set up to call VAMINIT to initialize each volume, build the PVT entry from the corresponding SDAT entry, and to examine the PAT table for relocated entries, setting PVTFLG to '80' if any are found. All assigned pages which are not in use are set available in the PAT. A SYSER (minor software) is invoked if a null field, unmounted volume, or more than three volume fields are discovered in a JFCB appendage. A return code of '10' is issued for the first two conditions and '14' for the third.

After all volumes are initialized, a check is made to determine if there are data set volumes that have not yet been mounted. If unmounted volumes remain, another DSCB must be read to obtain the next volume fields. If the "next" DSCB pointer is zero, a SYSER (minor software) and an ABEND (comp code 1) are invoked. Otherwise the READDSCB subroutine is entered to read the next DSCB (if necessary). A branch is made to the previously described code to process the volume entries in this DSCB. DSCBs are read until all volumes of the data set have been processed and the PVT has been built.

Once this has been accomplished, MOUNTVOL will check to see if any volumes had been specified in the JFCB. If not, the PVT pointer is moved to the return parameter list and a return is made to the caller with a code of '00'. If there were volumes specified, the volume serial numbers in the JFCB will be compared to those from the DSCB(s). If the serial numbers do not compare, they will be placed in the JFCB in the sequence defined by the user, but immediately following those placed there from the DSCB(s). If the volume serial numbers do compare, they will be ignored. The new volumes are then mounted, initialized, and entered into the end of the PVT via normal processing; return is made to the caller with a code of '00' and the PVT pointer in the third input parameter word.

MOUNTVOL also contains the READDSCB subroutine that is branched to when a DSCB is to be read. The READDSCB subroutine is entered with register 1 pointing to the DSCB to be read, and a word at location MNTDSCB that points to an aligned page of virtual storage (a page-boundary buffer) into which the DSCB page is to be read.

READDSCB stores the DSCB address for the READWRIT routine (CZCEM) and then puts the address of the page-boundary buffer into the parameter list for READWRIT. READDSCB assumes that the address of the JFCB is already in the parameter list. READDSCB then calls READWRIT to read in the DSCB page.

When READWRIT returns to READDSCB, the return code in register 15 is checked. If the return code is zero (good), READDSCB calculates the address of the format E DSCB and puts it into register 6. READDSCB then branches back to MOUNTVOL with register 6 pointing to the requested DSCB. If, upon return from READWRIT, READDSCB finds a non-zero return code in register 15, the return code is saved and then examined to see if it indicates a DSCB checksum error. If it does, READDSCB issues the following message:

CHECKSUM DID NOT COMPARE UPON READDSCB.

READDSCB then sets a return code to indicate a bad return from READWRIT and checks to see if the volume has already been mounted. If it has, then READDSCB calls RELEAS (CZCAD) to release the volume. Upon return from RELEAS, READDSCB returns to MOUNTVOL with the return coce (F'28') indicating a bad return from READWRIT in register 15.

MTREQ Routine (CZCAA)

MTREQ is a reenterable, recursive, privileged routine residing in virtual storage, used to allocate unit-record devices, and devices such as disk or tape drives required for private volumes. For a nonconversational task, the routine tries to allocate every device needed for that task. For a conversational task, it allocates the requested device as soon as it becomes available. (See Chart CA.)

Entry Points: CZCAA1 - Type I linkage. Entry is made under any of these conditions:

1. During conversational task execution.
2. During nonconversational task execution.
3. Prior to nonconversational task execution, that is, entry from the SECURE command routine.
4. When a specific device, rather than a type of device, is specified.
5. To exchange reservations when BUMP discovers the volume requested is presently mounted on another device (Nonconversational).

CZCAA3 - Entry at this point is for the purpose of resetting lock bytes when a task is abnormally ended or when SYSERR 080502509 occurs.

Input: On entry register 1 contains a pointer to one of the following 4-word parameter lists:

Entry Point 1 (private devices)

Word 1 Number of parameters minus one

Word 2 Pointer to a flag word

byte 0 - bit 0 - No msg to user
bit 1 - Mount all volumes in JFCB
bit 2 - JFCB pointed to by word 4
bit 3 - Dev Code pointed to by word 4
bit 4 - SDA pointed to by word 4
bit 5 - This is a reserve request
bit 6 - Scratch volume
bit 7 - CE is caller

byte 1 - Tape density code

byte 2 - not used

byte 3 - bits 0-2 not used
bit 3 - label status (unlabeled=1)
bit 4 - RING flag set (RECOGNITION=1)
bit 5 - tape ring required (RING=1)
bit 6 - ASCII request
bit 7 - do not verify volume label on call to PAUSE

Word 3 Pointer to a volume serial field or the number of device codes for reserve requests

Word 4 Pointer to a JFCB, to a list of device codes, or to an SDA

Parameters 3 and 4 may be repeated for device code or SDA requests.

Modules Called:
MSGWR (CZAAD2) -- To issue messages to a conversational user.

PAUSE (CZCAC1) -- To issue volume mounting messages to the operator.

AWAIT (CEAP7) -- To place the task in wait status.

GETMAIN (CZCGA2) -- To obtain more virtual storage.

FREEMAIN (CZCGA3) -- To release storage obtained by GETMAIN.

RELEAS (CZCAD1) -- To free any device that has been reserved but could not be used because of a non-zero return from PAUSE.

(CZCAD3) -- To update tables and/or release devices.

ADDEV -- To add symbolic device address to the Task Device List.

SETAE -- To set asynchronous entry for SAM modules.

RMDEV -- To remove the symbolic device address from the Task Device List.

PAIR (CZACS) -- To interface with the ABEND routine for the release of SDAT locks.

PRMPT -- To inform the user that he has attempted to exceed his device ration.

Exits: The routine normally returns to the caller, via the RETURN macro. If a minor system error occurs, the routine exits to ABEND. In the event that a major system error occurs the routine calls SYSER.

MTREQ checks for the following errors, placing a hexadecimal code in register 15 before returning control to the caller:

Code	Significance
00	No error detected
04	Volume not found
08	User canceled request
0C	Label error detected
10	Invalid device code or SDA
14	Private volume count exceeds limit
18	No room in request queue
1C	No reservation for this request
20	Volume requested is public
24	Volume-Id not found in SDAT
28	Flag incorrect in SDAT
2C	Volume in use
30	Error return from MSGWR
34	SDA requested device detached, partitioned or system reserve
38	CE requested partitioned device
3C	Ration exceeded
40	Non-zero return code from PAUSE

MTREQ also issues three messages to the user:

D301 NO PRIVATE DEVICE AVAILABLE FOR volume AT THIS TIME WILL YOU WAIT?

D302 PRIVATE VOLUME volume IS IN USE - WILL YOU WAIT?

CZCAA100 devcie NOT AVAILABLE - YOUR PRIVATE DEVICE RATION IS USED UP

OPERATION: For a conversational task, MTREQ allocates requested devices one at a time, as each becomes available. Allocation and volume mounting are done at once.

For a nonconversational task, MTREQ actually operates twice. First it is called by the SECURE command routine to reserve, as a set, all devices needed by the task. A nonconversational task cannot start until all of its required devices have been secured. Later it is called as each DDEF command is executed. MOUNTVOL calls MTREQ for VAM data sets, and it is also called if BUMP discovers the volume on another device. In SECURE processing, MTREQ reserves only private devices until the entire set is available; unit-record devices are reserved after all private devices have been reserved for the task.

If the caller furnishes a symbolic device address, and thus requests a specific device, that device will be allocated as soon as it becomes available.

When MTREQ allocates a device, it flags the symbolic device allocation table (SDAT) entry for that device as unavailable. Next it increments the TSI device queue, via ADDEV. Finally it calls PAUSE to carry out the volume mounting required.

When a required volume is mounted but not yet available, or if no device of the required type is available, or if the specific device requested is currently unavailable, MTREQ enqueues the request in the request queue (RQUE). A conversational user will be asked at this point if he wants to wait; he can cancel his request and end the wait at any time simply by pressing the ATTENTION key at his terminal. After queuing the request, MTREQ places itself in wait status which continues until RELEAS operates, allocates the device, and interrupts MTREQ. RELEAS will supply a pointer to the SDAT entry of the just-released device, and MTREQ then proceeds with its customary table and pointer update functions.

BUMP Routine (CZCAB)

BUMP is a reenterable, non-recursive, privileged routine, residing in virtual storage, used to dismount a specified private volume and mount another private volume on the same tape or disk drive. If the second volume is already mounted, device pointers are interchanged. The routine may be used just to reverify the label of an already mounted tape volume. (See Chart CD.)

Entry Point:
CZCAB1 - Type I linkage

CZCAB2 - Entry at this point is used to reset lock bytes when a task is abnormally ended.

Input: upon entry to this routine, register 1 points to the following parameter list:

Word 1	Address of first JFCB
Word 2	Address of volume serial field
Word 3	Address of second JFCB
Word 4	Address of second volume serial field

Modules Called:
Vol Label Processor (CZCWX) -- To read tape volume label for reverification.

PAUSE (CZCAC1) -- To issue mounting message to system operator and await operator response.

PAIR (CZACS) -- To provide for the release of SDAT locks when a task is abnormally ended.

RELEAS (CZCAD1) -- To release the original device if an exchange is made for a conversational task.

(CZAD3) -- Release for a nonconversational task.

MTREQ (CZCAA1) -- To reclaim the released device when an exchange is made for a nonconversational task.

Exits: BUMP returns control to the calling program, and sets a return code to show the results of BUMP operation. If a system error occurs, the routine is terminated by ABEND.

Return Code (Hexadecimal)	Significance
00	Normal return; no error detected.
0C	Reverification found incorrect label.
10	Old volume not mounted.
14	Device not available, that is, another task is using old volume.
18	Device codes differ but concatenation is not indicated.
1C	Reverification requested for a non-tape volume.
24	Device not tape or direct access.
28	New volume has already been mounted and is in use.

BUMP will also return any error code received from the PAUSE routine. See the

description of PAUSE for that routine's return codes.

Operation: BUMP begins by checking its input parameters to see if reverification is desired. This is indicated if the parameters (JFCB pointer and volume serial field) for both old and new volumes are the same. In this case, only a tape label check is made. BUMP sets a return code to show the result of the check before returning control to the calling program.

When a dismount/mount operation is requested, BUMP checks that the old volume is mounted, that the new volume is not mounted, and that the specified device is available. A comparison of the device codes for the two volumes decides the next action. If the volume is presently on another device, the pointers are exchanged.

If the device codes are the same (i.e., the new volume is to be mounted on the device used for the old volume), the routine builds a message to the system operator, asking him to mount the appropriate volume. BUMP then calls the PAUSE routine to transmit the message and wait for the operator's reply. If PAUSE reports a successful mount, BUMP updates the SDAT (symbolic device allocation table) entry for the device and adjusts the JFCBs to indicate that the old volume has been dismounted and the new volume mounted.

If the device codes differ and concatenation is specified, BUMP will try to mount the next volume in the concatenation that uses the same type of device. To do this, BUMP searches the concatenated JFCBs to find the next one with the same device code as the old volume. The JFCB and volume serial field of this volume are then accepted as the new volume parameters. The remainder of BUMP processing is the same as for volumes with identical device codes.

RELEAS Routine (CZCAD)

RELEAS is a reenterable, non-recursive, privileged routine residing in virtual storage, used to decrement the user count or, if the user count reaches zero, to release either a single device or all devices associated with a private data set. Release means to inform the system that the device upon which a private volume was mounted is now free for other use. The routine also notifies any task(s) awaiting the freed device(s) that the device(s) is now available. (See Chart CC.)

Entry Point:
CZCAD1 - To update the user count and/or release devices, and to calculate charges against a task for device utilization.

CZCAD2 - To reset lock bytes when a task is abnormally ended.

CZCAD3 - Same as CZCAD1 with no charges calculated.

Input: Register 1 contains a pointer to the parameter list:

Word 1 Pointer to a flag byte which contains one of the following:

X'80' JFCB pointer is given
X'00' SDAT pointer is given
X'40' SDAT pointer is given and drive is to be completely released.

Word 2 Pointer to a JFCB or SDAT entry

Modules Called:
VSEND (CEAQ5) -- To send message to task that was awaiting just-released device.

PAIR (CZACS1) -- To provide for the release of SDAT locks when a task is abnormally ended.

SETAE -- To set asynchronous entry.

PURGE -- To remove a device from a task's list of available devices.

Exits: RELEAS always returns control to the calling program, setting a return code to show the results of the RELEAS operation:

Return Code	Significance
X'00'	No error was detected.
X'04'	Release was for a public device; request was ignored.
X'0C'	Release was for device not assigned to this task; request was ignored.

Operation: RELEAS first inspects the input parameter list to see if an SDAT or a JFCB pointer was supplied. The parameter list consists of a pointer to a flag byte and a pointer to an SDAT entry or a JFCB (which resides in the task data definition table). An SDAT pointer is given when just one device is to be released. A JFCB pointer is given to release more than one device. The flag byte equal to '80' implies that the second pointer is to a JFCB.

When the pointer is to an SDAT entry, RELEAS reduces by one the user count in that entry. When the pointer is to a JFCB, the routine scans the volume serial numbers in that JFCB to see if any volumes are mounted. For each volume that is mounted, RELEAS reduces by one the user count in the corresponding SDAT entry, resets the

volume-Id field in the JFCB to the volume serial number, and zeros the volume mounted flag.

If the user count is now non-zero, no further action is taken. If an SDAT request, RELEAS merely returns to the calling program. If a JFCB request, the routine looks for the next mounted volume.

If the user count is zero, a PURGE is done to remove the device from the task's list of available devices. Then, if the release is being done during a nonconversational task, no further action is taken. Otherwise, the device is now free, and RELEAS searches the request queue for a request involving the device type just released. Priority is given to requests in this order:

1. Request for a specific device, by symbolic device address
2. Request from a conversational task
3. Any other kind of request

When a request can be filled, RELEAS sends a message to interrupt the task that made the request and supplies the appropriate SDAT pointer. Whether or not there is a request for the just-released device, RELEAS flags the SDAT entry for that device to show its availability.

PAUSE Routine (CZCAC)

PAUSE is a reenterable, non-recursive, privileged routine residing in virtual storage, used to send mount request messages to the system operator, asking him to mount volumes. It also verifies the operator's reply and, for tape or direct access, checks the label of the newly mounted volume. (See Chart CB.)

Entry Point:
CZCAC1 - Type I linkage.

CZCAC2 - Entry at this point is used to reset lock bytes when a task is abnormally ended.

Input: Upon entry to PAUSE, register 1 contains a pointer to a parameter list:

Word 1 Flagword address
The flagword is formatted as follows:
Bits 0-7 unused
Bit 8 volume verification
Bit 9 tape volume
Bit 10 scratch volume
Bit 11 unlabelled volume
Bit 12 no message to user
Bit 13 SAM volume
Bit 14 ASCII
Bit 15 unused
Bits 16-23 tape density
Bit 24 tape ring required
Bit 25 RING flag set
Bit 26 remount volume
Bit 27 reverify volume
Bits 28-31 unused

Word 2 SDAT entry address

Modules Called:
MSGWR (CZAAD) -- To issue messages to a conversational user.

SAM Vol Label Rdr (CZCWX) -- To rewind and unload a tape volume.

VSEND (CEAQ5) -- To send a message to the operator.

AWAIT (CEAP7) -- To place the task in wait status.

OBTAIN (CZCFO) -- To read a volume label and to read the VTOC DSCB or SAM organized volumes.

PAIR (CZACS) -- To provide for the release of SDAT locks when a task is abnormally ended.

BSAM READ/WRITE (CZCRA) -- To read a tape header label.

Tape Volume Label Reader (CZCWX) -- To read tape volume labels.

Exits: PAUSE always returns control to its caller, placing a hexadecimal code in register 15 to show the results of processing. The codes are as follows:

Code	Significance
00	No error detected.
04	Operator has made a negative reply, indicating he could not perform mount request.
08	Attention interrupt or notice of shutdown received.
0C	Label error or read error detected.

Operation: For a conversational task the routine next issues a message (via MSGWR) to inform the user that his task is now waiting operator action.

The mount request message, built by PAUSE, is sent to the operator via WTOR.

When PAUSE regains control, it checks first to see if an attention has occurred; if so it merely returns control to its caller after setting an appropriate return code. The action taken depends on the request. PAUSE now informs the user (pro-

vided he is conversational) that his task is no longer waiting, and returns control to the caller.

When verification has been stipulated, PAUSE proceeds according to the type of volume involved. The volume identification supplied by the caller (or by the operator, if a scratch volume) is compared against the actual volume label. For a disk, the routine copies the volume label. Some of the information from the volume label is used to update the SDAT entries designated by the caller. On SAM organized packs the volume label contains a pointer to the VTOC DSCB, which PAUSE calls OBTAIN to read. Some of the device constants and space information contained in the DSCB are also used to update the SDAT entry.

The density at which a new or existing tape is to be processed is always determined at mount time. The PAUSE routine uses the density specified in a newly defined field in SDAT (SDADN) for its label processing.

For labeled tapes, PAUSE attempts to read the volume label at the density specified in SDADN. If the label is not readable at the density specified in SDAT or if the label could not be read, a check is made to see if a scratch tape or user tape was requested. If a scratch tape was requested, the operator is instructed to mount another tape with the correct density. If a user tape was requested, PAUSE returns a "label error" return code (X'0C') to the calling routine. Once the volume label has been read successfully, an attempt is made to read header label 2, if any, at the density specified in SDADN. If there are header labels and the density specified in the header label does not agree with the density specified in SDAT or if the label could not be read, the same processing as for the volume label verification is performed. If there are no header labels and the volume label is read correctly, the density in SDAT is considered correct. The tape is always backspaced to its original position following the volume label, and processing continues.

If an unlabeled tape was requested, PAUSE attempts to read the volume label at all densities in order to verify that a label does not exist. If a label was found on a SCRATCH tape that was requested with no labels, the operator is asked to mount another tape. If a label was found on a user tape defined as unlabeled, PAUSE returns a "label error" return code (X'0C') to the calling routine. Note that the system cannot protect against the user specifying the wrong density for unlabeled, uncataloged user tapes.

For SCRATCH tapes only, PAUSE checks the expiration date in the first header label 1 on the tape to determine if the expiration date has been exceeded. The current date is obtained by use of the EBCDTIME macro instruction. If the header label 1 was properly read and the expiration date has been reached, or if the header label 1 did not exist or could not be read, the tape is backspaced to its original position following the volume label, and processing continues.

If the expiration date was not reached, a message is sent to the operator stating "EXPIRATION DATE NOT REACHED, (original mount message inserted here), OR REWRITE THIS TAPE BY REPLYING R." If the operator responds to rewrite the tape, a new expiration date is assigned and recorded in header label 1.

When the PAUSE routine requests the operator to mount a SCRATCH volume (labeled or unlabeled), it expects the operator to respond with the volume serial number of the volume mounted. If the volume is labeled and the operator's response does not match the volume serial number in the label, the operator is told to mount a new volume. If the operator's response is correct, the volume serial number is placed in SDAT. If the volume is not labeled the volume serial number supplied by the operator is placed in SDAT.

The PAUSE routine issues the following messages to inform the user of task processing:

B075 WAITING: YOUR VOLUME IS BEING MOUNTED

B076 VOLUME MOUNTED

B077 VOLUME SERIAL NUMBER number HAS BEEN ASSIGNED TO YOU

For a scratch volume that is not identified by the operator, the routine sends a request to the operator to provide the missing volume identification. When a volume cannot be verified, the operator will be prompted to inform him of the error. The number of times the operator message is issued depends on the limit set in system common by the installation. The message is:

ENTER SERIAL NO. OF VOL. ON (symbolic device address).

Other messages sent to the operator by the PAUSE routine are:

READ ERROR ON (symbolic device address). CORRECT IF POSSIBLE.

VOL. LABEL ERROR ON (symbolic device address).

Virtual Memory Allocation (VMA) provides a centralized routine for dynamically servicing all requests for virtual storage issued by the system or user's programs during the execution of a task. VMA determines if a request for virtual storage should be packed or placed in a unique segment, starting on a segment boundary. The general allocation requirements are for segmentation on a 32-bit system, and for packing on the 24-bit system. However, there is no firm division between the two systems, regarding the method of allocation used. Sharable data sets will always be located on segment boundaries, and private PSECTS will always be packed regardless of system type. In order that installations may choose other than the basic philosophy, and may arbitrarily allocate on segment or page boundaries, VMA is designed to allow implementation of either form of allocation on either system.

To provide VMA with this facility, the following parameters are presented to the system and are interrogated by VMA in the course of determining the pages to be allocated:

- System Packing Parameter: an indication of whether or not private control sections and private data sets should be packed.
- Public Segment Indicator: an indication of whether a public segment is being used.
- System Indicator: an indication of whether the system has 16 (24-bit addressing) or 4096 (32-bit addressing) segments.
- Virtual Storage Pointer: the virtual storage address at which packing can begin.
- Variable Length Indicator: Indicates variable or nonvariable allocation for next segment information.
- Next Available Segment Pointer: the virtual storage address of the next full segment available for allocation.
- Public Segment Number: the shared page table number of the public segment, if a public segment exists.
- Variable Allocation Parameters: the number of pages that must be added to the number of pages in a variable request.

These parameters, initialized in the System Table (SYS) or in the Interrupt Storage Area (ISA) -- page 0, segment 0 -- are copied into the PSECT of VMA the first time VMA is entered. In order to alter the basic algorithm to meet the need of a particular allocation, the GETMAIN macro may specify parameters which override, for that allocation, those which Startup presented to the system. The standard method of allocation is shown in Figure 6.

TYPE OF DATA	TYPE OF SEGMENT: PACKED	TYPE OF SEGMENT: NON-PACKED
Private CSECT	Packed Private Segment	Private Segment
Public CSECT	Packed Public Segment	Shared Segment
PSECTS	Packed Private Segment	Packed Private Segment
Private VAM Data Set (RESTBL)	Packed Private Segment	Private Segment
Public VAM Data Set (RESTBL)	Shared Segment	Shared Segment

Definitions:

Packed - Allocate on page boundary.

Private - Pages Tables are in the task's XTSI and are available only to this task.

Public - Page Table is in real core and is pointed to by a segment table entry for all tasks wishing to reference it. Packed by page boundary.

Shared - Page Table exists in real core and is pointed to by a segment table entry for all tasks who wish to reference it. Allocation is on a segment boundary.

Figure 6. Standard (Default) Virtual Memory Allocation

VMA has two primary functions. It controls the allocation of virtual storage and it controls the construction of segment and page tables by issuing the appropriate SVCs. The various control fields in the VMA PSECT provide a virtual storage picture of various aspects of the task's segment and page tables. On examining the nature of a request for virtual storage allocation, VMA (either private or shared) will examine its parameters under various algorithms, select the proper address for the allocation, update its parameters, and issue the appropriate SVC for updating the segment and page tables.

There are six entry points to Virtual Memory Allocation:

- GETMAIN (CZCGA2) - Get virtual storage by pages
- FREEMAIN (CZCGA3) - free virtual storage by pages
- EXPAND (CZCGA4)* - Expand an existing block of virtual storage
- GETSMAIN (CZCGA6)* - Get shared virtual storage
- CONNECT (CZCGA7)* - Connect to a shared page table
- DISCONNECT (CZCGA8)* - Disconnect from a shared page table

Note: Entry points marked * are available to privileged programs only.

Input and output parameters are discussed in the descriptions of the various subroutines.

VMA -- Virtual Memory Allocation (CZCGA)

The Virtual Memory Allocation Routine is a closed, re-enterable, privileged service routine located in the task's initial virtual storage. It is called via Type I or Type II linkage depending on the privilege class of the user. The primary functions of the module are to control the allocation of virtual storage, and to control the construction of segment and page tables by issuing the required SVCs. VMA is used by system routines that require storage space for a user's task as well as by user's programs.

Task interrupts (synchronous I/O, asynchronous I/O, external, timer) are inhibited on entry by use of an ITI (inhibit task interrupt) macro. The mask field is restored to its original state on return. (See Charts DA and DB.)

Entry Points:
CZCGA2 (GETMAIN) -- called via GETMAIN macro

(CZCGA3) FREEMAIN -- called via FREEMAIN macro

(CZCGA4) EXPAND -- called via CALL MACRO; Type I linkage

(CZCGA6) GETSMAIN -- called via CALL macro; Type I linkage

CZCGA7 (CONNECT) -- called via CALL macro; Type I linkage

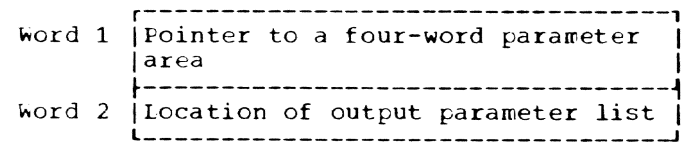
(CZCGA8) DISCONNECT -- CALLED VIA CALL macro; Type I linkage

Input: GETMAIN - register 0 contains:

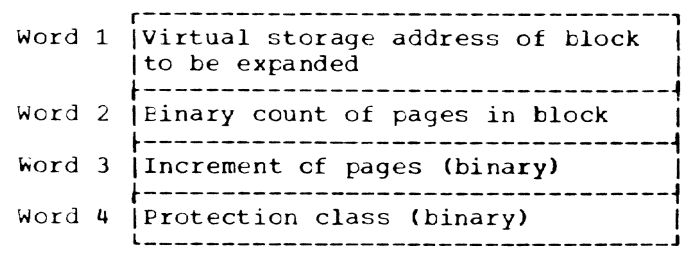
- Binary count of pages in lower three bytes
- Protection class (in binary) in bits 4-7 of the high-order byte
- Packing parameter (binary) in bits 1-3
- Variable parameter in bit 0

FREEMAIN - binary count of pages in register 0; variable parameter in sign bit of register 0; virtual storage address in register 1.

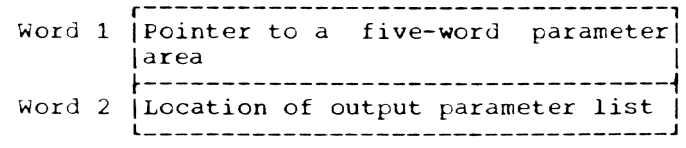
EXPAND - register 1 contains a pointer to the following two-word parameter list:



The four-word parameter area pointed to by Word 1 above is as follows:



GETSMAIN - register 1 contains a pointer to the following two-word parameter list:



The parameter area pointed to by Word 1 above is as follows:

Word 1	High order bit specifies EXIT=RETURN option if 1; remaining bits specify binary number of pages required.
Word 2	Shared page table (SPT) number (binary)
Word 3	Data type (binary)
Word 4	Protection class (binary)
Word 5	Variable length indicator

CONNECT - register 1 contains a pointer to the following parameter list:

Word 1 area	Pointer to a two-word parameter
Word 2	Location of output parameter list

The parameter area pointed to by Word 1 above is as follows:

Word 1	Shared page table (SPT) number (binary)
Word 2	Relative page location (binary)

DISCONNECT - register 1 contains the following:

Word 1	Pointer to a two-word parameter area

The parameter area is as follows:

Word 1	Shared page table (SPT) number (binary)
Word 2	Relative page location (binary)

Output: GETMAIN - Virtual Storage address of allocated block is returned in register 0

FREEMAIN - no output

EXPAND - the second pointer in the parameter list points to the word where EXPAND returns the virtual storage address

GETSMAIN - the second pointer in the parameter list points to the following parameter area:

Word 1	Shared page table (SPT) output
Word 2	Virtual storage address of allocated block

CONNECT - the second pointer in the parameter list points to a word where the virtual storage output is stored

DISCONNECT - no output

Modules Called:
ADDPG - builds page table and external page table entries

DELPG - deletes page table and external page table entries and releases main storage and external storage

MOVXP - moves page table and external page table entries from one location in XTSI to another

ADSPG - builds page table and external page table entries for shared pages

CNSEG - connects a segment table entry to a shared page table

DNSEG - disconnects a segment table entry from a shared page table

CKCLS - checks the privilege of pages being freed.

Exits:
Normal - registers 2-14 are restored

Error - VMA returns a code 8 in Register 15 if it receives a bad parameter or a code of 4 if insufficient virtual storage exists and the EXIT=RETURN option is specified. If the option EXIT=RETURN is not specified ABEND is invoked. An error exit is also taken when a nonprivileged user tries to free a page with a protection class other than user read/write, tries to free unallocated virtual storage, or in a 32-bit system, tries to free with a variable request a segment not marked as variable.

Flowchart References:
GETMAIN - DA
FREEMAIN - DB
EXPAND - DA
GETSMAIN - DB
CONNECT - DB
DISCONNECT - DB

GETMAIN (CZCGA2)

The function of GETMAIN is to obtain virtual storage for the user's program and system service routines (see Chart DA).

Entry Points: CZCGA2, Type IM or Type II linkage, via GETMAIN macro.

Input: Upon entry to this routine, general registers 0 and 1 contain input parameters as shown in Figure 7.

Parameter 1
count of contiguous pages requested by LV parameter of GETMAIN macro instruction

Parameter 2
packing parameter, as specified by PACK parameter of GETMAIN macro instruction

Parameter 3
protection class - specified by PR parameter of GETMAIN macro instruction

Parameter 4
specifies return code, as specified by EXIT parameter of GETMAIN macro instruction. Indicates if ABEND should be called if the request cannot be satisfied

Parameter 5
variable bit parameter

Modules Called:
ADDPG by macro.
CKCLS by macro.

Exits: Return, type IM linkage

Register 1 - contains the virtual storage address of the allocated virtual storage

Register 15 - contains one of the following return codes:

0 - request fulfilled;
4 - not enough virtual storage to fulfill request;
8 - invalid parameter passed to VMA.

Operation: GETMAIN examines the input parameters to determine the type of allocation required. If the number of contiguous pages is zero, a variable-length allocation is implied. In this case, the number of contiguous pages for the request is taken from the variable allocation parameter, initialized by STARTUP. In the 24-bit system, this amount is some number of pages less than 256 (20 if the variable allocation parameter is 0); in the 32-bit system variable segments are assigned. The user can also specify a variable allocation to GETMAIN through an input parameter. In this case, the number of pages in the variable allocation parameter is added to the number of pages requested and, for the 24-bit system, allocation of this sum is made. In the 32-bit system a full segment is added to the request and the allocation is indicated as variable to ADDPG.

Allocation occurs in one of the following two manners:

Packed: Allocation of packed virtual storage can be specified in either of two ways:

- If the system packing parameter is on; in the absence of an input packing parameter specifying a unique segment, virtual storage is packed.
- If the system packing parameter is off, and an input parameter to GETMAIN specifies packing, virtual storage is packed.

Non-packed: Allocated virtual storage is placed in a unique segment if the system packing indicator is off and the packing input parameter is not specified, or if the

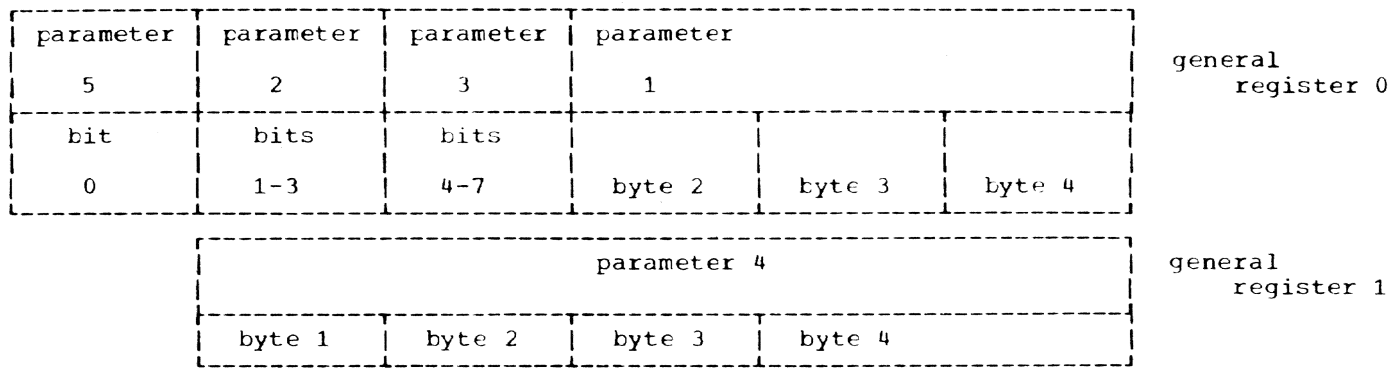


Figure 7. Location of Input Parameters for GETMAIN

input packing indicator specifies a unique segment.

Allocation of Packed Virtual Storage

In a 24-bit system GETMAIN first checks for freed pages to reassign. If the request is for eight pages or less, GETMAIN then checks to see if the request fits in the segment indicated by the virtual storage pointer. If it does, the block is allocated, and the virtual storage pointer is updated by the number of pages allocated. If the request does not fit, GETMAIN checks the segment map to determine if there is a segment available for reallocation. If a freed segment exists, GETMAIN allocates from the segment, starting at the segment boundary. The virtual storage pointer is set to the segment boundary plus the number of pages allocated, unless the entire segment was assigned. If no freed segment exists, GETMAIN checks to see if the next available segment is contiguous to the segment indicated by the virtual storage pointer.

If the segment is contiguous, GETMAIN determines if the request will fit in the space remaining from the virtual storage pointer to the end of virtual storage. If it does not, the user has exceeded the size of virtual storage, and ABEND is invoked. If it does fit, the virtual storage pointer is updated by the number of pages allocated. The next available segment pointer is updated if the allocation causes the virtual storage pointer to cross a segment boundary.

If the next available segment is not contiguous to the segment indicated by the virtual storage pointer, GETMAIN determines if the request will fit in the space remaining from the next available segment to the end of virtual storage. If it does not fit, the user has exceeded the size of virtual storage, and an error return is made. If the allocation does fit, the virtual storage pointer is set to the location of the next available segment plus the number of pages allocated.

In both of these cases, if the number of pages requested is greater than 256, no check for a freed segment is made. If the block cannot be allocated in any of the above manners, the page map, in the 24-bit system, is searched for a contiguous string of zeros for as many pages as are requested, and allocation is made at the corresponding virtual storage address. In the 32-bit system, the user has exceeded the size of virtual storage, and an error return is made.

Allocation of Non-Packed Virtual Storage

If the requested allocation can be contained in one segment, a check is made for a freed segment. If one exists, it is allocated, and its entry in the segment table is set to "1". If no freed segment exists, the next available segment is allocated, and the next available segment pointer is updated by one segment.

If the request cannot be put into one segment (i.e., the request is for more than 256 pages), the next available segment is checked to see if the request will fit in the space between the next available segment and the end of virtual storage. If it will, virtual storage is allocated there; if not, the segment map is searched for a large enough space to reallocate.

Allocation of Virtual Storage

When sufficient space is found, VMA issues the ADDPG supervisor call to create the page table and external page table entries. If the request is for more than 16 segments, the calls to ADDPG are broken into multiples of 16 segments. ADDPG sets the page table entries to "not available", the external page table entries are set to "assigned" and the file addresses are set to zero. ADDPG receives the protection class and sets the proper protection code in the external page table entries. GETMAIN returns the virtual storage address to the requestor.

If a request cannot be satisfied, the user has the option of retaining control or having GETMAIN go to ABEND by specifying the EXIT parameter. If this parameter is not supplied, GETMAIN goes to ABEND.

Restrictions: The requestor cannot assume that two consecutive requests for virtual storage will result in allocation of one contiguous area; this can only be done by issuing one request for the entire amount. A contiguous block of virtual storage, within the limits of the system can be requested; however, it must exist at the time of the request.

FREEMAIN (CZCGA3)

The function of FREEMAIN is to release virtual storage allocated through GETMAIN or GETSMAIN. (See Chart DB.)

Entry Point: CZCGA3, type IM or type II linkage, via FREEMAIN macro

Input: Upon entry, general register 0 contains a fullword count of the number of contiguous pages to be released; general register 1 contains the virtual storage address of the area to be released.

The variable allocation indicator is contained in the sign bit of register 0. If this bit is set to "1", a variable allocation is indicated; if it is set to zero, the request is treated normally.

Restrictions: Virtual storage obtained in pages must be freed by pages.

Modules Called:
DELPG - by macro.
CKCLS - by macro.

Exits:
Normal - Return via type 1M linkage.

Error - If a parameter is invalid, or if there is a protection violation, or if the storage to be freed is unassigned, an error exit is taken.

Operation: The operation of FREEMAIN depends on whether a variable allocation or fixed amount of virtual storage is to be released.

Release of a Variable Allocation

24-bit System: If the number of pages to be released specified as an input parameter is zero, release of a variable allocation is indicated. In this case, FREEMAIN releases the number of pages indicated in the variable allocation parameter. If the variable allocation indicator is on, a variable allocation is also specified to be released. FREEMAIN adds the number of pages specified in the variable allocation parameter to determine the number of pages to be released. The number defaults to 20 if the variable allocation parameter is 0. FREEMAIN uses the CKCLS supervisor call to determine the protection class of the virtual storage area to be released; it then determines if the user has sufficient privileges to release the area of virtual storage. (A nonprivileged user cannot release an area of privileged virtual storage.) If the user can release the area, it is released; if he does not have the privilege to release it, VMA will ABEND the task. FREEMAIN uses the virtual storage address and count of pages to update the page map.

32-bit System: If the number of pages specified in the LV parameter is zero, the release of a variable allocation is indicated. FREEMAIN indicates to DELPG that a variable-length segment is being released. The variable allocation indicator input to FREEMAIN also specifies the release of a variable allocation. FREEMAIN passes to DELPG the user count and indicates that the request is variable. DELPG returns page and segment counts to FREEMAIN so that an exact accounting of 'variable' pages can be made. FREEMAIN uses the CKCLS supervisor call to determine the protection class of the virtual storage area to be released, and to determine if, on a variable freemain request, the segment is variable.

Releasing Storage

FREEMAIN uses the DELPG supervisor call to free any auxiliary storage and main storage committed to these pages. DELPG sets the corresponding page table entries to "not available", the external page table entries to "not assigned", and the file addresses to zero. Except in the case of 32-bit variable segments, FREEMAIN divides all requests that overlap segment boundaries into multiple requests, so that each virtual storage request is contained within a single segment. DELPG supplies an output parameter that specifies the length of the segment after the deletion. In this way, FREEMAIN will update the segment map, enabling GETMAIN to reuse segments that were previously allocated.

DELPG packs the page tables and removes page table entries for a deletion at the end of a segment. Therefore, if the deletion is at the end of a segment, FREEMAIN moves the virtual storage pointer back whenever possible in the 24-bit system, and the next available segment pointer back in the 32-bit system.

EXPAND (CZCGA4)

The function of EXPAND is to enlarge an existing block of virtual storage. EXPAND can be used only by privileged system service routines. (See Chart DA.)

Entry Point: CZCGA4, type 1 linkage, via CALL macro.

Input: Upon entry to EXPAND, general register 1 contains a pointer to the following two-word parameter list; aligned on a fullword boundary:

Word 1	Virtual storage address of the four-word parameter area shown below
Word 2	Virtual storage address of where output parameter list is to be stored

Word 1	Virtual storage address of the block that is to be expanded
Word 2	N- a binary count of the number of pages in the above block
Word 3	N*- a binary count of the number of pages to be added contiguously to the above block

Word 4	Protection Class - a binary number, right-adjusted, representing the protection class of the block to be expanded. It can assume the following values: 0 - user read/write 1 - user read only 2 - private privileged

Modules Called:
CKCLS by macro.
MOVXP by macro.

Output Parameters: EXPAND returns the virtual storage address of the expanded block at the location specified by parameter 2 of the input parameter list.

Exits:
Normal - Return via type 1 linkage.

Error - If a parameter is invalid or if the user tries to expand unallocated virtual storage, an error exit is taken.

Operation: EXPAND receives the size of the original block of virtual storage, the size of the block to be added, the virtual storage address of the block, and the protection class of the block.

If the virtual storage pointer minus the size of the existing block is equal to the input virtual storage address, the added block can be appended to the existing block. The virtual storage address plus the size of the block are passed to a special GETMAIN entry to perform the allocation. The new virtual storage address is set up as an output parameter. Accounting calculations are performed, task interrupts are enabled, and return is made to the calling routine.

In the 24-bit system, if the virtual storage pointer minus N is not equal to the input virtual storage address, a check is made to determine if there is a hole of N* in the page map at the virtual storage address +N. If there is, the hole in the map is deleted, and GETMAIN is used as in the previous case.

In the 32-bit system (or in the 24-bit system when there is no available hole), EXPAND proceeds as follows.

The block must be relocated so a count of N + N* is passed to GETMAIN to obtain the block of storage. The supervisor call MOVXP is used to move the page and associated external page table entries to their new position. FREEMAIN code is then used to delete pages of the old block, task interrupts are enabled, and return is made to the calling routine.

In the 32-bit system, or in the 24-bit system with no hole of N+N* or greater in the page map, a user error has occurred; virtual storage is exceeded. A user error exit is made.

GETSMAIN (CZCGA6)

GETSMAIN is used to create a shared page table when one does not exist, and to obtain additional space in a segment when a shared page table has already been created. GETSMAIN is used by VAM and the dynamic loader to obtain shared virtual storage. (See Chart DB.)

Entry Point: CZCGA6 - via CALL macro. Type I linkage

Input: Upon entry, general register 1 contains a pointer to the following parameter list; two contiguous words aligned on a fullword boundary.

Word 1	Virtual storage address of the five word parameter area shown below.
Word 2	Virtual storage address, where output parameter is to be stored.

Word 1	N = binary number of pages required. N cannot exceed 1 segment (256 pages). When a request for pages cannot be filled, the high-order bit in this word indicates EXIT=RETURN option if bit is 1.
Word 2	SPT# - binary number of the shared page table when virtual storage is to be acquired. If this number is not known, an SPT# of binary 0 should be used, and a new unique number will be supplied by the system.
Word 3	Data type - one of the following numbers (in binary) with the following meanings: 1 - data set 2 - CSECT 3 - PSECT
Word 4	Protection class - one of the following numbers (in binary) with the following meanings: 0 - user read/write 1 - user read only 2 - private privileged

Word 5	Variable length indicator - one of the following numbers (in binary) with the following meanings: 0 - nonvariable allocation 1 - variable allocation

Exits:

Normal - Return, by macro. GETSMAIN returns the shared page table number at the location specified by the second parameter of the input parameter list. The full word following the SPT number contains the address of the allocated virtual storage.

Error - If a parameter is invalid or if the request for pages cannot be filled, a check is made to see if the high-order bit is on in the first word (N pages required) of the input parameter area. If this bit is off, an error exit is taken. If it is on, return is made to the calling routine with one of the following return codes in register 15:

X'04' Request cannot be filled
X'08' Invalid parameter received

Restrictions: The amount of shared storage requested, N, cannot exceed one segment (256 pages).

Modules Called: ADSPG by macro.

Operation: GETSMAIN disables task interrupts and obtains the input parameter.

If the variable length indicator input parameter is off (that is, set to "0"), GETSMAIN gets the next segment (which is either the next available segment, or the next available deleted segment). If the variable length indication input parameter is on (that is, set to '1'), the action taken depends on whether the system has 24-bit or 32-bit addressing. For the 24-bit system, GETSMAIN adds the pages indicated in the variable allocation parameter in the ISA to N (the number of pages requested); it then gets the next segment. In the 32-bit system, a request is made for a variable length segment, and this is indicated to ADSPG.

If the public segment indicator is off, (that is, a public segment does not exist), ADSPG is called; if this indicator is on, the type is tested. If the type is CSECT, the shared page table number of the last assigned segment is used as input to ADSPG; ADSPG is then called. If the type is not CSECT, the shared page table number is not known (except for the case where data sets are being packed into a segment and the shared page table number is known from the first GETSMAIN). When the shared page table number is not known, ADSPG creates a shared page table and sets up the corresponding segment and auxiliary segment table entries. When a shared page table number is supplied to ADSPG, ADSPG checks to see if the request will fit into the segment already assigned. In order to determine if ADSPG has used the segment supplied to it, GETSMAIN compares the shared page table number input to ADSPG with the shared page table number output.

If the request fits into the segment already assigned, the virtual storage address is set up as an output parameter (the second word of the output parameter list). Task interrupts are enabled, and an exit is made to the calling routine.

If the request does not fit into the segment already assigned, ADDSPG uses the virtual storage address supplied by GETSMAIN to create a new shared page table, segment, and auxiliary segment table entry. Therefore, in the case of packing within segments, it is ADSPG that keeps track of page allocation; however, GETSMAIN still keeps track of segment allocation.

GETSMAIN then saves the new Shared Page Table number and checks to see if the next segment was the next available segment.

If it was not, the next available deleted segment is deleted from the segment map, and the segment map is searched for a deleted segment less than the next available segment. If such a segment was found, the next available deleted segment is set to the found segment, the virtual storage address is set up as an output parameter, task interrupts are enabled, and return is made to the calling routine.

If the next segment was not the next available segment, a test is made to determine if it is the last segment, and if it is, it is set to "defunct", and its virtual storage address set up as an output parameter. After enabling interrupts, an exit is made. If the next segment is not the last segment the procedure is as stated above. The virtual storage address is placed in the output parameter list, task interrupts are enabled, and return is made to the user.

CONNECT (CZCGA7)

The function of CONNECT is to allocate shared virtual storage when a shared page table already exists for the data object. (See Chart DB.)

Entry Point: CZCGA7, type I linkage, available to privileged users only.

Input: Upon entry to CONNECT, general register 1 must contain the address of a parameter list. The parameter list, which must start on a full-word boundary, is as follows:

Word 1	Virtual storage address of the two-word parameter area shown below
Word 2	Virtual storage address where output parameters are to be stored

Word 1	SPT# - the binary number assigned to the shared page table to be connected
Word 2	Relative Page Location - the page number, in binary, within a shared segment of the beginning of the shared data object

Exits:
Normal - Return by macro.
CONNECT returns the virtual storage address of the shared data object at the location specified by the second input parameter.

Error - If a parameter is invalid or if there are no available segments, an error exit is taken.

Restrictions: Available to privileged users only.

Modules Called: CNSEG by macro.

Operation: CONNECT checks for the next available deleted segment; if found, this segment is passed as input to the supervisor call, CNSEG. If a deleted segment is not found, the next available segment is used. The shared page table number and relative page location are also passed to CNSEG.

If a segment is already connected to the shared page table, CNSEG notes the corresponding segment number and segment address and passes it back to CONNECT. If not, CNSEG creates a segment table and auxiliary segment table entry and connects them to the shared page table and passes the segment address to CONNECT.

If the segment returned from CNSEG is not the same one passed as input, CONNECT adds the relative location to the returned segment and sets it up as an output parameter, enables task interrupts and exits.

If the segment number returned from CNSEG is the same as was input to CNSEG, CONNECT proceeds as follows:

If the segment is the next available segment, CONNECT sets it to "defunct" if it is the last segment or increases the number of the next available segment by "1" if it is not the last segment. It then adds the relative location to the address of the returned segment and places the output parameter area. After enabling task interrupts, it exits.

If the segment returned is not the next available segment, the entry for the next available deleted segment is removed from the segment map, and the segment map is searched for a deleted segment less than the next available segment. CONNECT then sets the next available deleted segment to zero if no such segment was found, or, if such a segment was found, sets the next available deleted segment to the number of the found segment. CONNECT adds the relative location to the returned segment and places it in the output area. Task interrupts are enabled, and CONNECT exits.

DISCONNECT (CZCGA8)

The function of DISCONNECT is to unlink a task's segment table entry from an existing shared page table. (See Chart DB.)

Entry Point: CZCGA8, type I linkage, via CALL macro.

Input: Upon entry to DISCONNECT, general register 1 contains the address of the following parameter list

1 word aligned on a fullword boundary	Virtual storage address of a two-word parameter area

Word 1	SPT# - the number, in binary, assigned to the shared page table to be disconnected
Word 2	Relative Page Location - the page number, in binary, within a shared segment, at which the shared data object begins

Exits:
Normal - Return by macro.

Error - If a parameter is invalid an error exit is taken.

Modules Called: DSSEG by macro.

Restrictions: DISCONNECT is available for privileged users only.

Operation: DISCONNECT receives the shared page table number and relative page location as input parameters: nonzero relative page location should not be disconnected. If this segment is specified, DISCONNECT does not process the request. It merely enables task interrupts and exits. The public segment cannot be disconnected; therefore, DISCONNECT checks the shared page table number given as input against any public segment shared page table numbers. If the public segment is specified, DISCONNECT enables task interrupts and exits, as for the previous case.

If the segment specified can be disconnected, DISCONNECT uses the supervisor call DSSEG. DSSEG searches the auxiliary segment table, and when the corresponding entry is found, the segment is set to "not available", and auxiliary segment table is set to "not assigned". DSSEG supplies an output parameter specifying the virtual storage address of the segment corresponding to the input shared page table number. DISCONNECT updates the next available deleted segment and segment map accordingly, thereby making the segment available for reallocation.

Small Virtual Memory Allocation (SVMA) is used to assign or free virtual storage in multiples of doublewords expressed as a byte count. The routines may be used by all system service routines and users for dynamic virtual storage. Execution results from the R option of the GETMAIN and FREEMAIN macros.

GENERAL DESCRIPTION

Requests for virtual storage are made via the GETMAIN macro, R form. Requests to free previously assigned virtual storage are made via the FREEMAIN macro, R form. Requests for assignment of virtual storage must specify the number of bytes, and this value should represent a count of doublewords. If the request is not a multiple of eight (a doubleword multiple) the request value is increased to the next higher multiple of eight.

Virtual storage under control of SVMA is divided into two classes by type of user (privileged and nonprivileged) with a page header table for each type. This insures that the virtual storage of the system service routines is not available for use by user programs, nor can it be freed by them.

Each page of virtual storage controlled by SVMA is assigned a storage key. Requests for storage with a given key will be assigned areas within the same page if possible.

Requests are also classified into those for less than one page and those of one page or more. Requests for less than one page are aligned on a doubleword boundary, while those for a page or more are page-aligned.

Except for the program itself, all storage required for internal tables and allocation is obtained dynamically from GETMAIN (pages), thus the program only occupies the storage required for its operation.

The program uses three entries in the ISA table (CHAISA). These are:

ISAVPS
: Contains the address of this program's PSECT. It is not necessary for the user to supply the PSECT address.

ISALCK
: Task interrupt inhibitation lock byte. This is set whenever this program is called, then reset to its initial state on exit.

ISATMP
: Address of the Task Monitor's PSECT is used to determine whether the caller is the ENTER routine.

COMMENTS: The user should be aware that he may actually use more SVMA than he requests but he may then overlay this area with a subsequent get SVMA request. He may also be able to retrieve data after he has given a FREEMAIN SVMA request for the area.

Area that has been freed by the user will not be reallocated.

Internal Tables

There are three internal tables: (1) Page Header Table; (2) Page Headers; (3) Unit Table.

The Page Header Table Page represents pages of virtual storage. Each page is one entry in the table. Each page header table page has forward and backward links, a count of active page headers, and up to 51 page headers.

The page headers consist of 20-word entries within the Page Header Table. A page header contains all the information about a page of virtual storage which is assigned or available for assignment, including a Unit Table.

The Unit Table is used only by the FREEMAIN routine, and consists of 512 bits - one bit for each doubleword of the page defined by the Page Header.

The Entry Definitions for the Page Header Table are as follows:

PHT	Page Header Table
FORPAGE	Forward link - points to next Page Header Table in chain.
BACKPAGE	Backward link - points to last Page Header Table in chain.
PHCOUNT	Count of active page headers - those currently in use.
PHEADERS	Page headers - 20 words.

BACKPH — Backward link - points to last page header in chain.

FORPH — Forward link - points to next page header in chain.

PASK — Page address - (20 bits) address of the first byte of a page available for or currently in use by user; and Storage key - (12 bits) The storage key assigned to the page.

NAU — Next available unit - The address of the next available doubleword within the page (when combined with the page address it forms the address returned to the user).

NFU — Number of freed units - a count of the number of bytes which have been freed via FREEMAIN. This will equal the NAU address when all assigned bytes on the page have been freed.

UNIT — Unit Table - 512 bits initially zero. When FREEMAIN has freed a unit the corresponding bit is set to one.

Picture of Page Header Table

Name	Operation	Operand
PHT	DSECT	
FORPAGE	DS	F
BACKPAGE	DS	F
PHCOUNT	DS	0CL80
PHEADERS	DS	F
FORPH	DS	F
BACKPH	DS	F
PASK	DS	F
NAU	DS	H
NFU	DS	H
UNIT	DS	CL64

Internal Subroutines Available

PHTBLINK
This subroutine is used to connect one page header table page to the last one in the chain. It calls GETMAIN (pages) requesting one page with a storage key which is user-fetch-protected. It then inserts the backward link address and zeros the forward link address, the page header counter, sets the next available page header address to the address of the first page header in this entry, updates the forward link address of the last page header table (pointed to by the backward link), and returns.

PHLINK
This subroutine zeros the page header to be linked in, inserts the backward link address, updates the forward link address of the last page header (pointed to by the backward link), sets the storage key required, obtains a page, inserts the address in the entry, calls NAPHR and returns.

NAPHR
This subroutine locates the next available page header. It first searches the page header table page in which the next page header was located. If there are no page headers available, it then searches all other page header table pages for an available page header. If none can be found it sets the next available page header address (NAPH) to zero and returns. If one is found it inserts the address into NAPH, updates the page header count for this page header table page, and returns.

DELINK
This subroutine removes a page header from the chain. It updates the forward and backward pointers of all headers involved, subtracts one from the page header count, calls PHTDE if the count reaches zero and then returns.

PHTDE
This subroutine disconnects a page header table page from its chain. However, before disconnecting the entry it checks to see if there is only one page and if so it sets NAPH to point at the first page header and returns. If there are other pages it updates all the forward and backward links involved. Then if the next available page header (NAPH) points to a header in this page, the NAPHR routine is called to obtain an address in an existing page. The page is then released via FREEMAIN (pages) and control is returned to the main program.

SMALL VIRTUAL MEMORY ALLOCATION (CZCHA)

SVMA is used with GETMAIN macro R option, and FREEMAIN macro R option.

For GETMAIN macro R option:

This routine is a closed, reenterable, privileged service routine used by any system service routine or user to assign virtual storage in multiples of doublewords.

Entry Point: CZCHA2 (See Chart EA.)

Input: On entry Register 0 contains:

- The byte count in the low order 3 bytes.
- The protection class (0, 1, or 2) in bits 4-7 of the high order byte.
- Packing parameters in bits 1 to 3 of the high order byte.

Output: Register 1 contains the Virtual storage address and/or Register 15 contains one of the following codes:

00 Normal return. Successful completion.

04 Insufficient contiguous virtual storage available, if EXIT = RETURN.

08 Protection class is invalid.

Internal Subroutine Used:
PHTBLINK - Page Header Table Page Link.

PHLINK - Page Header Link.

NAPHR - Next Available Page Header Routine.

Modules Used:
GETMAIN (CZCGA2) To supply integral pages.

ABEND (CZACP) To force abnormal end of task.

Exits:
Normal - Return to calling routine with Register 15 containing 00 and Register 1 containing the virtual storage address.

Error - If EXIT = RETURN option has been used, control is returned to the user and Register 15 contains a code of 04.

If EXIT = RETURN and not enough virtual storage exists, ABEND is invoked.

If the protection class is invalid, a code of 08 is returned in Register 15.

Restrictions: Two separate requests for virtual storage do not assure contiguous allocation of the two areas. The only way to ensure a contiguous allocation of N pages or bytes is the GETMAIN of N pages or bytes.

Operation: When a request is made, GETMAIN saves the present value of ISALK and sets it on. This inhibits task interrupts while GETMAIN is in execution. The type of user is then determined by comparing the calling routine's PSECT address with the PSECT address of the ENTER routine (CZCUE). If it is the ENTER routine the user is nonprivileged; if not, the user is considered privileged.

The request is then adjusted to the next higher multiple of eight, if not on a multiple at present. If the request is for a page or more, a flag (MULTI) is set on.

The program then searches the proper page header table (privileged or nonprivileged) for a page with a storage key which matches the requested key. If, in this search, it is found that all the page headers have been used (that is, the next available page header field (NAPH) is zero), the subroutine PHTBLINK is used to connect another page header table page into the chain and to initialize it. If NAPH is not zero, but a new page header is needed, the subroutine PHLINK is called to link a new page header into the chain.

The search can end in four ways: (1) A page is found which has enough storage available to satisfy the request. The next available unit (NAU) address is saved for output to the user and the NAU address is then advanced by the amount of the request. The return code (in Register 15) is set to zero and the page and unit address put in Register 1. All program switches are then reset, the ISAICK value is restored, and control is returned to the caller. (2) The request is for greater than a page so that one page cannot satisfy the request. GETMAIN (pages) is called to supply enough pages to fulfill the request. The NAU address of all full pages is set to 4096. The program then handles the fractional page (if one was requested) as described in 1, with the exception that the address returned to the user is the address of the first page assigned to the user. (3) There is no page which has enough storage available. GETMAIN (pages) is called to supply one page and the program proceeds as described in (1). (4) There are no pages which have enough available storage and GETMAIN (pages) cannot supply the required page (or pages). The program will either return control to the calling routine with Register 15 set to 4 (if the EXIT = RETURN parameter was used), or will call ABEND.

SVMA does not attempt to back-fill pages. This means that if FREEMAIN (bytes) has freed a portion of a page, the areas will not be reassigned until all units of the page which were assigned (via GETMAIN bytes) are freed. The page is then released (via FREEMAIN pages), allowing reassignment of the page. The NAU address, the number of freed units, and the unit table are all set to zero.

For FREEMAIN macro R option:

This routine is a closed, reenterable, privileged service routine used by any system service routine or user to free virtual storage in multiples of doublewords.

Entry Point: CZCHA3 (See Chart EB.)

Input: On entry:

- Register 0 contains a binary count of the bytes to be released.
- Register 1 contains the virtual memory address of the bytes to be released.

Output: Register 15 is returned to the user with one of the following codes:

00	Normal return. Successful completion.
04	Address to be released cannot be located, is unassigned, or has been freed.
08	Address to be released is not doubleword address.

Internal Subroutines Used:
DELINK Delink a page header.
PHTDE Page Header Table Delink.

Modules Used:
FREEMAIN (CZCGA3) To free integral pages.

Exits:
Normal - Return to the calling routine with Register 15 containing 00.

Error - If address to be released cannot be located, is unassigned, or has been freed, control is returned to the calling routine with a code of 04 in Register 15.

If the area to be released is not a doubleword address, control is returned to the calling routine with a code of 08 in Register 15.

Operation: When a request is made, FREEMAIN operates in the same manner as described for GETMAIN to save ISALCK, determine the type of user, adjust the requested byte count, and set MULTI (one page or more indicator). The VMA is checked for a doubleword boundary. If it is not a doubleword boundary, a code of 08 is returned to the user.

The routine then searches the correct page header table (privileged or nonprivileged) for the page header which has the specified page address (high order 20 bits of the input address). This search ends when the page header controlling the page is located. If the page header cannot be found, a code of 04 is returned in Register 15.

Once the correct page header is located, the program checks that the next available unit address is not less than the maximum address to be freed. This is an error condition, unless the request was for more than a page, and results in a return to the caller with a code of 04 in Register 15. If the request exceeded one page, then the page being freed is checked to insure that all units were assigned and that no units were previously freed. If either of these conditions is not met, an error is indicated and control is returned to the caller with a code of 04 in Register 15. If these conditions are met then the page is removed from the chain via the DELINK subroutine, the request is decreased by 4096, the starting address (address of first unit to be freed) is increased by 4096, and the search is re-started for the next page involved. This process continues until the request value is reduced to one page or less. When the correct page is located this time, or if the initial request was for less than a page, and the next available unit address is not less than the maximum address to be freed, the program checks to insure that all units to be freed are now assigned (the unit table contains 0's in each unit position to be freed). If all units are not assigned (one or more contains a 1 bit), an error condition exists and control is returned to the caller with a code of 04 in Register 15. However, if all units are assigned, the program frees them by placing 1-bits in each position to be freed.

The next step is to determine if all units are now free for that page. This is done by comparing the next available unit (NAU) address with the number of freed units. If they are equal, the page is freed via FREEMAIN (pages) before beginning the normal return sequence. Any pages previously processed to be freed are freed via FREEMAIN (pages), Register 15 is set to 0 (in case of an error resulting in a return code in Register 15, these two operations are bypassed when returning), program switches are reset, all other registers are restored and control is returned to the user.

A symbolic library is a collection of separate components called parcels, each parcel being a named group of symbolic statements, which are combined into one VISAM line data set and indexed so that any single parcel may be retrieved by specifying its name. The parcels of a symbolic library may be macro definitions for use by the TSS/360 Assembler, or they may be any other groups of statements which the user might wish to store within the system and retrieve during the execution of his programs.

Associated with the symbolic component of each symbolic library is a separate data set called the index which contains in alphabetic order the names of all the parcels in the symbolic component and the line number of the first symbolic statement in each named parcel. Once the symbolic component and the index have been created, system programs (such as the assembler) or user programs may call a system routine to automatically search the index for a given parcel name and thus locate the desired parcel to be processed.

The symbolic component of a symbolic library may be created by the user using the system DATA command, and may be modified using the system MODIFY command. Alternatively, the user may employ any program that constructs a VISAM line data set. Once the symbolic component has been created, a system routine may be called by the user to automatically construct the index for the entire library. If the library is replaced or modified it may be automatically re-indexed after each such modification, by invoking the indexing program via an appropriate RUN command.

The following routines are invoked by the user to index and retrieve symbolic libraries:

Symbolic Library Indexing Routine (SYSINDEX)

This routine automatically indexes the symbolic component of a symbolic library to create an alphabetical index of all the parcels, and is invoked when the user issues the appropriate RUN command or executes a program calling this routine. Before execution, DATADEF commands must be issued for both the symbolic component and indexed data sets. The ddname SOURCE and dsorg (data set organization) VI must be specified in the DATADEF for the symbolic component. The DATADEF for the indexed data set must specify the ddname INDEX and dsorg of VS. The individual parcels are identified by SYSINDEX either automatically or by means of a user-supplied subprogram.

In the automatic method, parcels are identified by being immediately preceded by one or more header lines, which are distinguished by having a unique first byte distinct from the first byte of any symbolic line. Each header contains one name or alias for the following parcel. The parcel name (or alias) begins with the second byte of each header and may have at most the number of bytes specified by the user in his LENGTH parameter (see below). Alternatively, the user may employ any means for identifying parcels and their names, and may supply a subprogram to examine each SOURCE line provided by SYSINDEX and pass the name, aliases and retrieval line number for each parcel back to SYSINDEX.

The SYSINDEX routine analyzes the user-supplied parameters and calls the actual index-building routine SYSXBLD, which scans the entire SOURCE data set, testing each line for the user-supplied header identifier byte or allowing the user-supplied subprogram to perform the test and extract each name and alias. SYSXBLD collects a list of the name and aliases of each parcel in the temporary array LIST until the first non-header line is detected; then the names in LIST are placed one by one into a temporary index TINDEX along with the retrieval line number of the first non-header line in the parcel.

The temporary index TINDEX is used in order to sort the entries into EBCDIC collating sequence in the final index CHASLX. Each TINDEX entry consists of the fields (name, retrieval line number) which will be entered in the final index, plus a forward link pointing to the relative location of the TINDEX entry for the next higher name in EBCDIC sequence. As each name is received for entry into TINDEX it is compared with all previously entered names, in ascending sequence, until its proper location in the ascending chain is determined. Then it is placed at the end of TINDEX with a link field pointing to the next higher name, and the link field of the next lower name is adjusted to point to the new entry.

Once the entire SOURCE data set has been scanned and the complete TINDEX created, the last phase of SYSXBLD extracts all the entries from TINDEX in ascending name

sequence and places them, without the link field, into the final index CHASLX.

Symbolic Library Search Routine (SYSEARCH)

This routine may be called by a system or user program to locate any individual parcel of a symbolic library, using its index as created by SYSINDEX. The virtual address of the index and the name of the parcel are given to SYSEARCH, and the retrieval line number of the first line of the parcel is returned. The calling program should, before calling SYSEARCH using a given index for the first time, open the data set containing the index, find the index member and GET the index in order to give its address to SYSEARCH.

It is the function of SYSEARCH only to locate the line number beginning a parcel; the calling program must then access and process the parcel as required. Once the retrieval line number of a particular parcel is obtained and the data set containing the symbolic component is open, the calling program may use the SETL macro instruction to position the symbolic data set to the beginning of the parcel and then get each line of the parcel in turn until the end of the parcel is detected. The assembler, for example, may detect the MEND symbolic statement as the end of the parcel; a user program might either use a special symbolic line within the parcel or detect the next parcel header as the delimiter of this parcel.

User Subroutine for SYSXBLD

The user subroutine for scanning the symbolic statements, if provided to SYSINDEX, will be entered by SYSXBLD after each SOURCE line is obtained, and will be supplied with the virtual address of the last source line, the address of the DCB for the SOURCE data set, and the addresses where a retrieval line number (in 8-byte EBCDIC format) and a name or alias may be returned to SYSXBLD by the user. The user subroutine may inspect the line and return one of the following condition codes to SYSXBLD:

0 - No index entry to be made; proceed to get next SOURCE line

4 - Enter the returned name and retrieval line number into TINDEX and return to user subroutine without getting another source line

8 - Enter the returned name and retrieval line number into TINDEX and proceed to get next SOURCE line

If there are one or more header lines (not considered part of the parcel) containing the name and aliases of the following parcel, then before returning either code 4 or 8 the user routine must have located the proper retrieval line number to be associated with the following parcel by repeating the VAM GET macro instruction until the first line of the parcel is obtained; the DCB address supplied to the user routine must be used in this macro. During this process, the name and any aliases must be accumulated internally by the user routine until the proper retrieval line number for the following parcel is available to be passed back to SYSXBLD. Return code 4 is then used repeatedly to pass back all but the last alias and return code 8 is used to passback the last (or only) name for entry into TINDEX. The user routine is always entered at the same point and must therefore remember when returning code 4 not to re-process the last line obtained by SYSXBLD.

When the SOURCE data set is exhausted, whether it is SYSXBLD or the user routine issuing the last GET macro, control automatically passes to the last phase of SYSXBLD for constructing the final index.

SYSINDEX -- Symbolic Library Indexing Routine (CGCKA)

SYSINDEX receives the user's input parameters, prompts the user for missing parameters, and processes the parameters. After the parameters have been processed, SYSINDEX calls SYSXBALD, the routine that actually builds the index (CHASLX). (See Chart FA.)

Entry Point: SYSINDEX

Input: Parameters are received in one or more control statements in the form:

keyword=parameter, keyword-parameter

Keyword	
LENGTH	Number of characters in each name to be entered in the index
HEADER	Single character that is compared with the first byte of each source line to determine whether that line requires an index entry
SCAN	The symbolic name of a user-supplied routine to be called for each source line to determine if the line is an index entry

Note: HEADER or SCAN, but not both, must be specified.

Output: Symbolic library index - a virtual sequential data set. The index contains the names of, and line number pointers to, the elements contained in the source data set.

Restrictions:

1. If the user is operating in nonconversational mode, all input parameters to SYSINDEX must be correctly spelled in a single control statement.

2. Names to be placed in the index must not be longer than 255 characters.

3. An input parameter must not be split between two control statements.

4. Continuation records are not permitted.

Assumptions:

1. Before SYSINDEX is executed, the user must have stored the symbolic component as a line data set, organized sequentially by line number.

2. The user must have supplied two DDEF commands: one named SOURCE that defines the data set containing the source lines to be processed, and one named INDEX that defines the index to be created.

Modules Called:
VAM
GTWAR
GATWR
SYSXBLD-CGCKB - to create the index CHASLX

Exits:
Normal - end of job

Error - if user is nonconversational, any parameter errors cause the index generation process to be terminated and a return to SYSIN for the next command

Operation: Using the GTWAR macro instruction, SYSINDEX requests the user to enter his input parameters. Records received with errors are not processed. The SCAN routine (described below) scans the record for a delimiter (equal sign, comma, or blank). The first scan obtains a keyword; the second, a parameter; the third, a keyword; and the fourth, a parameter.

If the length operand is specified as zero or is not specified, the conversational user is prompted to supply the length, execution is terminated for the nonconversational user. If a valid length is specified, it is converted to binary and stored.

SYSINDEX then checks that either the HEADER or SCAN keyword (but not both) has been specified. The header character or address of the scan routine is stored.

When SYSINDEX determines that either HEADER or SCAN and a non-zero length have been specified, it calls SYSXBLD to generate the index. Upon return from SYSXBLD, SYSINDEX returns to the Command Language Interpreter.

If the user makes an error in entering his parameters or omits a required parameter, the GATWR macro instruction is used to write the appropriate error message.

SCAN SUBROUTINE: SCAN is an internal subroutine used exclusively by SYSINDEX. It scans the user's control statement to extract keyword symbols and parameter values. The scan detects delimiters, which may be: comma, equal sign, or an EOR marker. Imbedded blanks, which also halt the scan, are discarded. SYSINDEX and SCAN allows blanks to surround delimiters, but not within a keyword symbol or a parameter value. From its normal exit, the SCAN routine provides the character field scanned (left-adjusted with trailing blanks), the length of this field, and the delimiter. Delimiters obtained by SCAN are retained in a storage area until replaced by the next delimiter. When SCAN detects that the delimiter of the last field scanned was an end-of-record marker, all information on the control statement has been processed, and SCAN makes an end-of-record exit.

SYSXBLD -- Build Symbolic Library Index (CGCKB)

SYSXBLD consists of two major components: LOOP, which scans each line of the source data set to locate, extract, and place entries in a temporary index; and TERM, which operates on the temporary index to form a final and complete index. (See Chart FB.)

Entry Point: SYSXBLD

Input: Register 1 contains a pointer to this list:

Word 1	Pointer to full word containing length of parcel names in index
Word 2	Pointer to one-byte field containing header character, if specified
Word 3	Pointer to variable-length field (with maximum of eight bytes) containing name of user's scan routine, if specified

Output: The index CHASLX to be used in the subsequent retrieval of information from the symbolic library.

Modules called:
VAM
FINDJFCB
OPEN
FIND
GET
PUT
STOW
CLOSE
User's scan routine (if scan option is specified)

Exits:
Normal - return to calling program

Error -

1. If the same name is specified more than once, SYSXBLD terminates.

2. If an unreadable record is encountered in the source member, control is transferred to the SYNAD exit. SYSXBLD provdies no SYNAD exit for the CHASLX DCB; any unrecoverable I/O error associated with CHASLX invokes the system error procedure.

Operation: FINDJFCB is used to determine the nature of the symbolic component; it is assumed that this data set has been described by a DEFINE DATA command with the ddname SOURCE. The data set is opened and a FIND is issued for the member. GETMAIN obtains a page of virtual storage for the temporary index. The VISAM GET macro instruction (locate mode) is used to obtain a line from the symbolic component. If an unreadable record is encountered, control is transferred to the I/O error routine (IOERR) by means of the SYNAD exit in the DCB; GATWR is used to write an error message, and exit is made. Upon normal exit from GET, the input parameters are checked to determine the option selected - scan or header.

If scan was supplied, a CALL is made to the named routine. If the user's scan routine determines that no entry is to be placed in the index for a given line, it returns with a code of 0. If a name and line number are to be placed in the index, the AETI (add entry to index) subroutine is alled to supply the appropriate entry to TINDEX. A return code of entries 4 or 8 is returned.

If the return code is 4, SYSXBLD returns to the user's scan routine to obtain a synonym for the entry. This process is repeated as long as the user's scan routine returns a code of 4. (Note that it is the user's responsibility to avoid an infinite loop here.) The user must also make certain that the correct retrieval line number pointer is returned in the parameter list. This retrieval line number must be that of the first symbolic statement to be input following a CALL to SYSEARCH for a given name. Consequently, it is the first symbolic line of a parcel following the line containing the parcel name; the user's scan routine must get the former line to obtain the appropriate line number. For this reason, and so that the user can obtain synonyms already in the symbolic component, the DCB location is given in the CALL to the user's scan routine.

If the user's scan routine returns a code of 8, SYSXBLD makes the appropriate entry in TINDEX, gets the next source line, and then calls the user's scan routine as described in the previous paragraph.

If the user supplied a header character in the CALL to SYSXBLD, the character is compared with the first byte of the symbolic line. When an equal comparison is made, the name is placed in LIST; this name begins in the second byte of the symbolic line. The length of this name is given in the CALL to SYSXBLD. Note that the names are not placed in the index at this time, since this header may be immediately followed by other headers carrying synonymous names.

Entering of names in the index is postponed until a non-header is encountered, so that the correct retrieval line number may be obtained. When a non-header line is encountered, the list of names is dequeued, and AETI places the names in the index. GET then obtains the next source line.

This process is repeated until there are no more symbolic lines to process; the ECDAD exit in the DCB associated with the SOURCE data set is then taken. The SOURCE DCB is closed, and the DCB associated with the index is opened. FINDJFCB is used to determine the nature of the index. Members of partitioned data sets are located by the FIND macro instruction, which also indicates if the member is new or a replacement; this influences the type of STOW macro instruction to be used later. For replacement members, a SETL macro instruction is issued to position to the beginning of the index. PUT (locate mode) is then used to obtain the address of this index.

The length of the entry is extracted from the operand field of the CALL macro instruction. LINK, a link pointer to the next-higher entry in the index, is established. The entries are placed in the index until LINK attains a value of zero, indicating that the highest value name has been reached. Processing is then complete.

AETI SUBROUTINE: The add entry to index routine (AETI) is an internal subroutine used exclusively by LOOP. AETI shares the PSECT used by SYSXBLD. It expects a parameter list containing the location of the name to be an index and the location of the associated retrieval line number.

Entries are made in a temporary index (TINDEX), not in the final index. Entries in TINDEX are ordered physically in the order of submittal, but are linked together by ascending EBCDIC collating sequence. TINDEX entries contain:

1. A left-adjusted parcel name (NAME), the length of which was specified in the CALL to SYSXBLD.

2. The retrieval line number (RLN) contained in an 8-byte EBCDIC field.

3. A link pointer (LINK) to the highest name in ascending EBCDIC collating sequence contained in a 4-byte field. The highest value name has a zero link.

AETI maintains NEXT, a pointer to the first unused byte in TINDEX. Since the size of TINDEX is not predictable, the GETMAIN macro instruction is used, as needed, to acquire additional working storage.

SYSEARCH -- Symbolic Library Search Routine (CGCKC)

The SYSEARCH routine is used to locate information stored in a symbolic library. (See Chart FC.)

Entry Point: SYSEARCH

Input: Register 1 contains the address of this list.

Word 1	Address of the index component of the library to be searched.
Word 2	Address of first byte of name to be located. This name must be of the length specified to SYSINDEX or SYSXBLD and must be left-justified with trailing blanks.
Word 3	Location at which SYSEARCH is to store the retrieval line number it obtains.

Output: If the index entry is found, SYSEARCH places the retrieval line number at the location specified by word 3 of the input parameter list and returns with a code of 0. If the name does not exist in the index, return is made with a code of 4.

Modules Called: None

Exit: Normal - return to calling routine

Operation: SYSEARCH executes a conventional binary search using the starting point (SLXSSP) established by SYSINDEX or SYSXBLD.

DELTA, the increment value used in the binary search, is initially set to a value of SLXSSP-4+SLXNLN. SLXNLN is the length of a line in the index. The input name is compared with the name retrieved from the index. If an equal comparison is made, the line number is returned to the calling program. SYSEARCH exits with a return code of 0.

If an unequal comparison is made, DELTA is divided by 2 (by a shift-register instruction) and added to, or subtracted from, the current position pointer (register 15), depending on whether the routine must move forward or backward in the index to find the desired name. The current position pointer is initialized to the binary search starting point.

The input name is compared to the index name indicated by register 15. If the input name is greater, SYSEARCH moves forward in the index; if the input name is less, SYSEARCH moves backward. Register 15 is adjusted and the comparison repeated.

On an equal comparison, SYSEARCH places the retrieval line number associated with the input name at the location indicated by word 3 of the input parameter list, sets the return code to 0, and exits. If DELTA reaches a value of SLXNLN+8, the input name does not exist in the index. SYSEARCH exits with a return code of 4.

The Control Section Store routine processes user requests made through the CSTORE macro instruction. During program execution, any set of contiguous virtual storage bytes may be transformed into an object module consisting of a single control section. The module is stowed in the current JOBLIB. It can then be loaded by the program that created it, or by a subsequent program. When the module is loaded, no relocation takes place; therefore, it may contain no relocatable items.

The resulting module will consist of an unnamed control section which contains a copy of the hexadecimal text beginning at the page boundary preceding the address specified as the starting address parameter, and terminating at the page boundary following the address computed from the fourth parameter. Thus the resulting control section will always be an integral number of pages in length.

When the module is loaded by the user, the module name, as well as the entry point name, will point to the address computed by adding to the address of the new module the page offset (if any) implied by the starting address. For example, assume that the user requests that a control section of 4048 bytes be created from the bytes beginning at virtual storage address 5D050. Also assume that the new module is later loaded at 70000. The loaded module and control section will occupy the full two pages beginning at 70000. The second page is required so that the new control section will include the last two bytes requested by the user. The module and entry point names will both point to 70050.

Maximum control section size is one segment.

CONTROL SECTION STORE (CZCKZ)

The Control Section Store routine is a privileged, reenterable, public system routine which permits the user to create a control section during program execution. (See Chart GA.)

Entry Point: CZCKZ, type I or II linkage

Input: General register 1 contains a pointer to the following parameter list:

Word	Contents
1 and 2	Module Name
3 and 4	Entry Point Name
5	Virtual storage address of first byte of text
6	Length of text, in bytes
7	Control section attribute code, as follows: Bit 24 on - System Bit 25 on - Privileged Bit 26 on - Common Bit 27 on - Prototype (PSECT) Bit 28 on - Public Bit 29 on - Read-only Bit 30 on - Variable length Bit 30 off - Fixed-length

Restrictions: The module name and entry point name must be unique in the job library. The control section to be created must not be more than a segment in length.

Modules Called:

CZCLA (OPEN)	
CZCLB (CLOSE)	To detect duplication and
CZCOJ (FIND)	to place the created
CZCOK (STOW)	module in the library
CZCOS (PUT)	

Exit: Return, with register 14 containing the exit address, and register 15 containing one of the following return codes:

00 - normal return

04 - module name or CSECT name already in use.

Operation: This routine receives input parameters passed to it by the CSTORE macro instruction. After opening the DCB for the current job library, it checks for potential duplication of the module and entry point names by issuing a FIND. If duplication exists, the DCB is closed, the return code in register 15 is set for duplication (X '04'), and the routine exits.

The PMD is now constructed in this routine's PSECT work area. The def for the CSECT is given a name of eight hexadecimal zeros, and V- and R- value displacements of zero.

The offset between the address given as the third parameter and the preceding page boundary is installed as the V- and R-value displacements in the def for the entry point name. The ref for the module name (standard entry point) is set to name this def.

The new PMD and the specified text are placed in the current JOBLIB with successive calls to move mode PUT.

The STOW routine is then called to enter the specified module name in the POD as the member name, and the specified entry point name as an alias.

The DCB is closed, the register 15 return code set to zero, and the routine exits.

The serviceability aids consist of routines recording error information, routines retrieving error information, and a time conversion routine, SYSTIME, which converts time from machine format to EBCDIC time and date.

ERROR INFORMATION RECORDING AND RETRIEVAL

During the course of TSS/360 operations, a history is maintained of the environment of the system at the occurrence of any hardware failures or major software errors. The pertinent information for each error -- error indications, machine status information, instruction retry data -- is collected by various TSS/360 programs and recorded on the paging drum for later retrieval and analysis by the Customer Engineer.

The paging drum is formatted so that following every page of 4096 bytes there is an unused record of 246 bytes into which error statistics may be placed. The error environment information is recorded on the even numbered records on each track. The first of these records (track 0, record 2) contains a pointer to the end of the error information on the drum, as well as some summary information. Approximately 192,000 bytes, capable of storing information about 500 error incidents, are available per paging drum. If additional incidents occur after the recording area is full, further recording will be bypassed until a retrieval program is run, freeing the recording area.

ERROR INFORMATION RECORDING

Machine checks resulting from central processor and storage unit hardware detected errors, system errors, and solid outboard errors on direct access devices are accumulated and recorded on the paging drum by the system error recording and retry (SERR) program.

The virtual memory statistical data recording (VMSDR) and virtual memory error recording (VMER) programs are called when a task I/O retry operation for a sequential device either ends successfully (intermittent error) or with error after a prescribed number of retries (solid error). VMSDR accumulates intermittent error information on the statistical data table (SDT), and calls VMER to record error information on the paging drum when a SDT field overflows or in the case of a solid outboard error. VMER is also called by SAM posting, TAM posting, MSAM posting, and RTAM reporting interface routines in the event of a solid inboard error.

ERROR INFORMATION RETRIEVAL

Two programs have been developed for retrieving the information from the drum, organizing the data into a useful format, and sending it to an output device. The first of these, virtual memory environment recording edit and print (VMEREP), runs as a virtual storage program under the Time Sharing System. The second program, environment recording edit and print -- Model 67 (EREP67), is a stand-alone program operating in real core. These retrieval programs are reserved for the use of the customer engineer.

VIRTUAL STORAGE I/O OPERATION AIDS

Two I/O operation aids are fundamental to virtual storage error recording and retrieval: the I/O request control block (IORCB) and the I/O statistical data table (SDT).

I/O Request Control Block (IORCB)

The IORCB is the basic I/O communication link between virtual storage and the resident supervisor. A virtual storage program requests the execution of an I/O operation by issuing the LOCAL supervisor call. Following the LOCAL is a variable length parameter list termed the IORCB.

The IORCB is variable in size, but the first 80 bytes are considered to be a fixed length sub-area whose internal fields are all in fixed and known locations relative to the beginning of the IORCB. There are three variable sub-areas:

- Data buffer
- Page list
- CCW list

The page list has a maximum size of eight doublewords. The maximum collective size of the three sub-areas is 1840 bytes.

I/O Statistical Data Table (SDT)

The I/O statistical data table (SDT) accumulates statistical data on outboard failures of task I/O devices. The SDT contains one statistical data record (SDR) entry for each task I/O device on the system. Each of these SDR entries consists of statistical data on outboard failures of the associated task symbolic I/O device.

Fields in the SDT are:

SDTLSD - Length of an SDR entry (72 bytes)

SDTLBA - Address of byte following last byte of SDT

SDTSDA - Symbolic device address

SDTFB - Flag bytes (Example: write-to-operator flag)

SDTLP - Last path used (actual I/O address)

SDTEIC - Total error incident count

SDTRET - Total retry count

SDTRTH - Retry threshholds (specific error condition for device)

SDTTS1 - Time stamp at error n-2

SDTTS2- Time stamp at error n-1

SDTTS3- Time stamp at error n

SDTSDB - SDR save area (64 half-bytes)- a 4-bit frequency counter for each bit of sense data

If an SDT field (bucket) overflow occurs, the SDR entry of a symbolic I/O device is written on the drum for preservation recording.

Virtual Memory Statistical Data Recording (CZCRY)

The purpose of the virtual memory statistical data recording (VMSDR) subroutine is to accumulate error statistics on task I/O devices in the statistical data table (SDT). VMSDR is a privileged, reenterable subroutine operating in virtual storage, and is called by SAM posting, TAM posting, MSAM posting, and RTAM reporting interface routines when a task I/O retry operation either ends successfully (intermittant outboard failure) or is completed with error after a prescribed number of retries (solid outboard failure). (See Chart HA.)

Entry Point: CZCRYY, type-1 linkage, via CALL macro.

Input: Register 0 contains the following error codes:

1 - intermittent outboard error.
2 - solid outboard error.

Register 1 contains a full word pointer to the failing CCW address in the IORCB.

Modules Called:
CZCRX1 - Virtual memory error recording (VMER) - generates I/O error records to be output on drum via drum access module (DRAM), and informs operator of the failing task I/O component if the immediate report flag is on for the device.

CZABQ1 - Write to operator (WTO) - transmits a message to the main operator control program (MOCP) which will print the message on the operator's terminal.

Exits:
Normal - when recording is completed, control is returned to the caller via a RETURN.

Error - if the statistical data record (SDR) entry for the failing device cannot be found in the SDT, a SYSERR of the minor software error type is invoked, followed by an ABEND macro.

Operation: VMSDR is called by the posting routines to record hardware outboard errors on I/O devices. The posting routines distinguish between two types of errors; solid and intermittent. VMSDR recognizes four types of errors; solid, immediate report, critical intermittent, and statistical data recording.

A solid error (call type 2B) is any error which was not recovered from after a specified number of retries by the access method. Intermittent errors are those which have been recovered from successfully.

If the immediate report flag is on in the SDT entry all intermittent errors for that device will be recorded as immediate report errors (call type 261). If the immediate report flag is not set, a test is made to determine if the error is any one of five critical errors on direct access. Critical direct access errors are recorded as intermittent errors (call type 2F).

If the error is neither a solid nor critical error and the immediate report flag is off, the statistical data table

entry for the device will be updated and tested for bucket overflow.

Updating the SDT table consists of recording the current system time if this is the first error incident (indicated by a zero error incident count) and incrementing the error incident count by one. The 64 sense bits are checked individually and the corresponding SDT bucket is incremented by one if the bit is on. There are 64 buckets of a half byte each. An overflow indicator is set whenever a bucket value reached 15. In the event of bucket overflow an SDR error (call type 2A) will be recorded.

If there is no overflow VMSDR sets the retry count in the SDT entry to zero thus completing the update of the statistical data table. VMSDR then returns control to the posting module.

If the error was solid, immediate report, or intermittent, or if there is a bucket overflow condition, the retry count remains in the SDT and the last path is moved from the IORCB into the SDT. The parameter list is completed and a call is made to VMER to record the error on the drum. On return from VMER a test is made for error type. If the error was either an immediate report or SDT type, the SDT buckets and error incident count are reset. The retry count is then reset and a return is made to the posting routine.

Virtual Memory Error Recording (CZCRX)

Virtual memory error recording (VMER) is a privileged, reenterable, subroutine, operating in virtual storage, which informs the operator of the failing task I/O component and generates I/O error records that are to be output for preservation recording on drum via the drum access module (DRAM). VMER is called by VMSDR in the event of a solid outboard failure or an SDR bucket overflow, and by SAM posting, TAM posting, MSAM posting, and RTAM reporting interface routines on a solid inboard failure of a task I/O operation. (See Chart HB.)

Entry Points: CZCRX1 - Entry from virtual memory statistical data recording (VMSDR) to form the I/O outboard error record (CHADER) and reset to zero the SDR buckets in the SDT entry. Type-I linkage, via CALL macro.

CZCRX2 - Entry from SAM posting and TAM error posting routines to form an I/O inboard error record. Type-I linkage, via CALL macro.

CZCRX3 - Entry from task monitor to call DRAM for recording I/O error records on drum. This is a queued linkage entry.

CZCRX4 - Entry from Main Operator Housekeeping Routine (MOHR) to initialize the drum index if it is invalid.

Input: Upon entry at CZCRX1, register 1 points to a parameter list which contains the following:

Word 1 - SDT entry address of failing component.

Word 2 - Failing CCW address (zero if unpredictable).

Word 3 - Sense bytes 0-3.

Word 4 - Sense bytes 4-7.

Word 5 - One of the following error codes:
X'00' - call type 2B solid error
X'08' - call type 26 immediate report error
X'10' - call type 2F intermittent error
X'18' - call type 2A SDA error

Word 6 - Address of SDAT entry for failing component.

Upon entry at CZCRX2, register 1 points to a parameter list that contains the following:

Word 1 - Address of failing CCW (Zero if unpredictable).

Word 2 - Pointer to failing symbolic address.

Upon entry at CZCRX3, register 0 contains 22 minus the buffer number to be recorded. Register 1 contains the address of the page containing the buffer.

Output: If VMER is entered from VMSDR, and if the immediate report flag is on, the following message is printed on the operator's terminal for all solid outboard errors:

I/O OUTBOARD ERROR ON SDA XXXX,SOLID

For all other outboard errors the message is:

I/O OUTBOARD ERROR ON SDA XXXX,I.R.

The variable is the symbolic device address field taken from the SDT entry for the device in error.

If VMER is entered from a posting routine, and if the immediate report flag is on, the following message is printed on the operator's terminal:

I/O INBOARD ERROR ON LP XXXX

The variable is the last path used taken from the SDT entry for the device error.

Modules Called:
CZASY - Drum access module (DRAM) - outputs I/O error records on the dummy spaces on the paging drum.

CZABQ1 - Write to operator (WTO) - transmits a message to the main operator control program (MOCP) which will print the message on the operator's terminal.

CZGG2 - GETMAIN routine used to allocate space for buffering data which is to be recorded.

CZGG3 - FREEMAIN routine used to release buffer space when no longer required.

CZCJTQ - Queue linkage entry routine used to set up later entry at CZCRX3 by task monitor.

Exits:
Normal - (After entry at CZCRX1 or CZCRX2) When Queued Linkage entry has been set up, VMER returns to caller via RETURN.

(After entry of CZCRX3) After I/O error records have been recorded on drum, VMER returns to caller via RETURN.

Error - If an I/O error occurred while outputting the I/O error records on drum, the following message is printed:

I/O DRUM FAILURE ON SDA XXXX

If no available drum path can be located, the following message is printed on the operator's terminal:

NO DRUM PATH AVAILABLE

If the drum used for recording is full, the following message will be issued:

DRUM OVERFLOW ON SDA XXXX

Operation: When VMER is called by VMSDR to format and record task outboard errors, the code in word five of the parameter list is used to set the proper error type in the outboard error record. Error information is moved from the SDT entry for the failing task I/O device and from the IORCB in the interrupt storage area and the ISA to the outboard error record. An in-line routine is then entered to determine the proper CCW list, failing CCW, number of CCWs, and OER record length all of which are then stored in the outboard error record. If the failing CCW is unpredictable all of the CCWs (up to a maximum of 10) are saved in the error record. If the failing CCW is known and its relative number is 10 or less, all the CCWs (up to a maximum of 10) are saved along with a pointer to the failing CCW. If the failing CCW is known, but its relative number is greater than 10, the number of CCWs and relative number of the failing CCW are both set to 10 and the failing CCW and the nine CCWs preceding it are saved in the outboard error record.

If the failing device is tape or disk, the volume ID and the current system time are put in the outboard error record. Current system time reflects the time of the last error for an SDR record or the time of error for the other types of task outboard errors.

When VMER is called by posting routines to record task I/O inboard errors (call type 2C) the time of error is put in the inboard error record and an immediate report message is set up for the operator. The CSW, Channel log, user ID, last path, last seek address, and SDA are stored in the inboard error record. If the failing CCW is known the CCWs are moved to the inboard error record.

The remaining procedure is the same for both types of entry (CZCRX1 and CZCRX2). If the immediate report flag is on, the WTO macro is invoked to transmit the message to the main operator control program which prints the message out on the operator's terminal.

If there is no buffer space available, the GETMAIN routine is called to allocate a page for use as buffers. The I/O error information to be recorded is moved into an available buffer space and a queued linkage entry is set up to provide a letter entry by task monitor at CZCRX3. The parameters required to locate the buffer are also provided. VMER then returns to caller via RETURN.

Task monitor then calls VMER at CZCRX3 and passes the parameters used to locate the buffer containing the I/O error information to be recorded.

A CCW list is constructed for a channel program to locate and read in the environment recording (ER) index record. The CCW list is described in the drum access module (DRAM) description. The drum access module (DRAM) subroutine is then called to execute the channel program.

Upon return from DRAM, the return code is checked to determine if the file protect bit is on (set to one), indicating that the task-supervisor interlock is locked or the task-task interlock is locked for another

Condition Code	Error	Recovery Procedure
1	SIO instruction reject	Generate message to operator informing him of drum I/O failure.
2	Operation complete with error	Issue RDI and return to caller.
2	Drum Path unavailable	Get new drum SDA from first half word of parameter list and repeat recording sequence. If new SDA is zero, issue RDI and inform operator that no drum path is available.

Figure 8. DRAM Condition Code Recovery Procedures

task. If the file protect bit is on, the sequence of operations beginning with the construction of a CCW list is repeated. If not, the condition code is checked to determine if the I/O operations have successfully completed. If successful, a check is made to see if the time stamp exists. If there is nc time stamp, the current system time is obtained via the REDTIME macro and stored in the index time stamp. If the index record is invalid, it is initialized and the current system time is stored in the error index time stamp.

DRAM is then called to read the first available 246 byte dummy record from the drum. The error record is moved from the error buffer to the dummy record.

If the entire record did not fit in the dummy record, DRAM is called to write the dummy record onto the drum and the record number portion of the index is updated. The remainder of the error record is then moved to the drum buffer and DRAM is called again to write the next dummy record on the drum.

If the entire record fits in the dummy record which was read, DRAM is only called once to write the dummy record on the drum and the record number is left unchanged.

The byte count portion of the index record is updated and DRAM is called to write the index back onto the drum.

If the condition code indicates that I/O operations have not been successfully completed, one of the following recovery procedures is taken, depending upon the condition code returned by DRAM. See Figure 8.

When the I/O error record has been recorded on the drum, VMER resets the task interlock by issuing the extended instruction RESET DRUM INTERLOCK (RDI). If the buffer which was recorded was the first buffer in the buffer page, the page is released by calling the FREEMAIN routine. VMER then returns to the caller via RETURN.

When VMER is called by the Main Operator Housekeeping Routine (MOHR) during the startup process, entry point CZCRX4 is used. A flag is set to indicate entry from MOHR, and a branch is made to the routine normally entered at CZCRX3. The drum index is read, and the current system time is stored in the index record. The updated index record is written on the drum, the drum index interlock is reset, and control is returned to MOHR.

Drum Access Module (CZASY)

The drum access module (DRAM) is a special purpose, privileged, closed, reenterable, virtual storage subroutine provided for the use of those virtual storage programs, such as VMER and VMEREP, which must access the error records stored on the dummy spaces of the paging drum. This special access method is needed primarily to prevent interfering with the drum paging operations. (See Chart HC.)

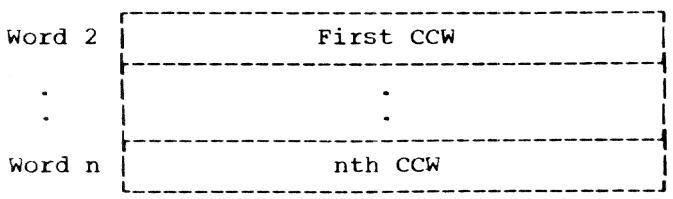
Entry Point: CZASY1 - Type-1 linkage, via CALL macro

Input: Register 1 contains a pointer to a location containing a pointer to a parameter list which contains the following:

byte	0	1	2	3
Word 1	Device Address	Flags*		CCW Count

*bit 2 of the Flag Byte contains the "ignore sick unit" indication.

The remaining input parameter list is in the form of a CCW list:



CCW n must be a NOP VCCW.

Output: Register 1 is returned to the calling program unchanged; the parameter list is unchanged, except for the following changes in Word 1:

Flag Byte:

Bit(s)	Meaning
0	File protect was on.
1*	Device address has been changed.
6,7	Condition Code after SIO:
	-00- Operation complete
	-01- Error after SIO
	-10- Complete with error
	-11-* Drum unavailable

* In the event that condition code 3 has been received, a new device address is returned in bytes 0 and 1 of Word 1.

Modules Called: IOCAL (SVC) - called to initiate the I/O operation.

Exits:
Normal - when I/O operations are complete, DRAM RETURNs to caller.

Error - when I/O operations are either complete with error or incomplete, DRAM RETURNs to caller with condition code indications.

Operation: The calling routine provides DRAM with a CCW list, which is used to build an I/O request control block (IORCB). A page list for the IORCB is constructed by inserting the high-order 20 bits of each CCW address into the list. The origin of the CCW list is calculated, and each output CCW is created relative to the page table. The length of the IORCB is then calculated in 64-byte blocks, and, when creation of the IORCB is complete, the I/O operation is initiated by executing an IOCAL SVC.

A special DRAM flag bit (IORAMM) is set in the IORCB to notify the page drum queue processor and other programs concerned that the call is from DRAM. When the I/O operation completes, the DRAM posting routine (CZASX) is given control. When posting is completed, control is returned to DRAM, which determines if a new device address was posted, as indicated in the condition code bits of the flag byte.

If a new device address was posted, the new one is placed in the first two bytes of the parameter list; otherwise, these bytes remain unchanged. The appropriate flags and CCW count are inserted, parameters replaced, and the registers are restored. DRAM then RETURNs to the calling program.

Virtual Memory Environment Recording Edit and Print (CZASE)

Virtual memory environment recording edit and print (VMEREP) retrieves, formats, and prints the environment recording information stored on the paging drum. VMEREP is a non-reenterable, privileged, virtual storage service routine reserved for the customer engineer. The routine receives job requests from the C.E. or a user with a privilege class of A or E, and communicates with him via the GATE subroutine (CZAAB). (See Chart HD.)

Entry Point: CZASE1 - Type-I linkage, via VMEREP command.

Input: Error history reports located on the dummy spaces on the paging drum.

Output: Formatted reports are sent to printer or SYSOUT.

Modules Called:
DDEF (CZAEA) - Obtains use of printer for output.

OPEN (CZCEA) - Opens output data set.

DRAM (CZASY) - Accesses error records on paging drum.

CLOSE (CZCLB) - Closes output data set.

RELEASE (CZAFJ) - Releases DDEF for printer.

GTWSR (CZATC) - Communicates with C.E. or user.

Exits: Normal - Exit is made when user enters job request END.

Operation: After identifying himself as a C.E. or user with proper privilege class, the C.E./user enters the command verb VMEREP which is interpreted by CLI. VMEREP is then brought into the system and given control. The C.E. then enters the following information via the console typewriter:

- Symbolic Device Address of the paging drum.
- Job Request.

If the desired output device is a printer, the VMEREP routine calls DDEF (CZAEA) to obtain a printer.

The job request specifies the type of information to be retrieved, and the options applicable. The following job requests and options are valid:

LIST FAILURES - Lists all errors recorded on the drum by record-id and error type.

PRINT INDEX - Print contents of the index record (track 0, record 2).

PRINT ALL - Print contents of the index record and all error information on the drum.

PRINT ID HHRRNN - This job request is given when information about a particular error is desired. HH, RR, and NN are the hexadecimal values of the track number, head number, and byte number respectively.

SEARCH CSEE - This request is used when a particular class of errors is to be retrieved and processed.

C is the CPU number (1-8) to which failure was atributed.

S is the storage element identification (A-H) to which failure was attributed.

EE is error type, as follows:

01 - Internal machine check.
09 - Multiple internal machine checks.
26 - Immediate report.
27 - Paging I/O statistical data record.
28 - Solid paging I/O outboard error.
29 - External machine check.
2A - Task I/O statistical data record.
2B - Solid task I/O outboard error.
2C - Task channel failure.
2D - Paging channel failure.
2E - Intermittent paging I/O outboard error.
2F - Intermittent task I/O outboard error.
41 - System error.

Any parameter not used should be replaced with an X.

Example: SEARCH XX29 means print all records whose error type is 29.

RESTART - This request causes VMEREP to return to its initial routines.

RESET INDEX - This request should only be used whenever all the data on the drum has been retrieved, so that programs which record information on the drum can start at the first record (track 0, record 4).

SET INDEX TO HHRRNN - The pointer in the index record is changed to HHRRNN. Any new records written on the drum will be recorded just after the byte located at track HH, record RR, byte number NN.

SET IR ****(SDA in four hexadecimal digits) - This job request sets the immediate report to operator flag in the SDT which causes each error for the particular device to be recorded.

RESET IR ****(SDA in four hexadecimal digits) - This request causes the immediate report flag to be set off.

END - This job request is used when no further requests are to follow, and indicates that data sets should be closed, devices relinquished, etc.

The following options are available to the user:

PRINT ZEROES - This option forces the printing of the complete CPU log, zeroes as well as non-zeroes.

Note: This option is applicable only to error types 01 and 09 and must be entered in one of these three ways:

```
SEARCH CS01 PRINT ZEROES
SEARCH CS09 PRINT ZEROES
PRINT ALL PRINT ZEROES
```

SYSOUT - This option causes the output of this job request to be sent to SYSOUT instead of the printer.

After the user enters his job request, VMEREP retrieves the desired information via the drum access module (DRAM), formats the data, and prints it on the specified output device. If VMER or SERR stored additional information on the drum while the servicing of the job request was in progress, and is applicable to the type of information requested, this new information is formatted and printed.

When the job requested has been completed, VMEREP asks the user for his next job request. When no more error data is to be retrieved, the user normally issues a RESET INDEX job request followed by an END request. VMEREP terminates via normal program end.

Environment Recording Edit and Print, Model 67 (CMASN)

The environment recording edit and print program for Model 67 (EREP67) is an off-line, self-loading, and self-controlled program used by the customer engineer to edit and print the information recorded on the paging drum by error recording programs

(SERR and VMER). This program does not run with the TSS monitor. It loads itself into storage and operates under its own control. (See Chart HE.)

Entry Point: The deck provided is self-loading. The program is assumed to be the sole user of the system at execution time.

Input: Error information stored on the dummy spaces on the paging drum.

Formatted reports are sent to the output device (usually a high-speed printer.

Assumptions:

- The TSS monitor will have to be stopped before this program can be used.
- The EREP67 program will have access to paging drum and a print device.
- EREP67 does not save or restore any environment.

Exits: Normal - exit is made when C.E. enters job request END.

Operation: EREP67 cannot be run concurrently with the TSS/360 monitor, since access to the paging device is required. After loading the program, the operator enters the following information via the console typewriter.

- Address of the paging drum.
- Address of the output device.
- Job request.

The following job requests, which specify the type of information to be retrieved and the options available, are valid:

PRINT INDEX - Print contents of the index record (track 0, record 2).

PRINT ALL - Print contents of the index record and all error information on the drum.

PRINT ID HHRRNN - This job request is given when information about a particular error is desired. HH, RR, and NN are the hexadecimal values of the track number, head number, and byte number respectively.

SEARCH CSEE - This request is used when a particular class of errors is to be retrieved and processed.

C is the CPU number (1-8) to which failure was attributed.

S is the Storage Element identification (A-H) to which failure is attributed.

E is the error type as follows:

01 - Internal machine check.
09 - Multiple internal machine checks.
26 - Immediate report.
27 - Paging I/O statistical data record.
28 - Solid paging I/O outboard error.
29 - External machine check.
2A - Task I/O statistical data record.
2B - Solid task I/O outboard error.
2C - Task channel failure.
2D - Paging channel failure.
2E - Intermittent paging I/O outboard error.
2F - Intermittent task I/O outboard error.
41 - System error.

Any parameter not used should be replaced with an X.

Example: SEARCH 3X01 means print all type-01 records pertaining to CPU 3.

PRINT ZEROES - This option is used in connection with the PRINT ALL, PRINT ID, and SEARCH job requests, to force the printing of the complete CPU log, zeroes as well as non-zeroes.

RESET INDEX - This request should only be used whenever all the data on the drum has been retrieved, so that programs which record information on the drum can start at the first record (track 0, record 4).

SET INDEX TO HHRRNN - The pointer in the index record is to be changed to HHRRNN. Any new records written on the drum will be recorded just after the byte located at track HH, record RR, byte number NN.

END - This job request indicates that no more job requests follow, and that the program is to be terminated.

After the C.E. enters his job request, EREP67 retrieves the desired information from the paging drum, formats the data, and prints it on the specified output device. When no more jobs are requested, the C.E. normally issues the RESET INDEX job request followed by an END request.

RTAM Error Recording Interface Module (CZCTR)

The RTAM error recording interface module (RERIM) is a privileged, reenterable, subroutine residing in virtual memory which acts as the central point for passing error information between the RTAM subsystem and the TSS subroutines VMSDR and VMER. The purpose of this interface routine is to allow terminal error recording information to be stored in virtual memory and not tie up real core space unnecessarily. RERIM is passed error information in the MCB messa-

gearea from the real core RTAM error routine CEATCS. RERIM then analyzes the information and passes it on to either VMSDR or VMER for recording. (See Chart HG.)

Entry Points: CZCTR1 - Entry from LOGON of Main Operator, and SHUTDOWN. When entered from LOGON this entry point is used to establish CZCTR2 as the entry point for processing external interrupts with code value 255. When entered from SHUTDOWN, this entry point is used to delete CZCTR2. This is a Type-I linkage entry point, entered via the CALL macro.

CZCTR2 - Entry from the task monitor for recording external interrupt 255 (X'FF'). This is a queued linkage entry.

Input: Upon entry at CZCTR1, register 1 contains a value of X'04' if called by SHUTDOWN.

Modules Called: CZCRYY - Virtual memory statistical data recording (VMSDR) - to accumulate error statistics on task I/O devices in the statistical data table (SDT) and to inform the operator of the device failure if the system is in the immediate report mode.

CZCRX2 - Virtual memory error recording (VMER) - to inform the operator of a solid inboard failure of a task I/O operation and record error records on drum.

Exits:
Normal - control is returned to the calling routine via a RETURN macro.

Error - if CZCTR2 receives control for an error other than a solid outboard, solid inboard, or intermittent error, a SYSER is issued and control is returned to the caller via a RETURN macro.
- if the SDA of the terminal is not found in CHBSDT a SYSER is issued and control is returned to the caller via a RETURN macro.

Operation: When CZCTR is entered at CZCTR1, register 1 is checked to see if the routine was called by LOGOFF. If it was, CZCTR1 issues a DIR macro to end CZCTR2 external interrupt processing and then returns to LOGOFF. Otherwise, CZCTR1 issues a SEEC to create an ICB and a message area for external interrupts. CZCTR1 then issues a SIR designating CZCTR2 as the external interrupt 255 handling routine and returns to the caller.

When CZCTR is entered at CZCTR2 an ITI macro is issued to inhibit interrupts. CZCTR2 then examines the error recording block in the message area defined with the SEEC macro issued by CZCTR1.

If the error is an intermittent or solid outboard error, VMSDR is called to record the error information. If it is a solid inboard error, VMER is called to record it. If the error is none of these, SYSER is called to indicate a system error. After recording the error information, CZCTR2 restores ISA fields so they contain the same information they had when CZCTR2 first received control. CZCTR2 then issues a PTI to permit interrupts and returns control to the task monitor.

TIME CONVERSION

SYSTIME ROUTINE (CZCTA)

The SYSTIME routine is a closed, reenterable, nonrecursive, virtual storage routine. It is used by the EBCDTIME macro to convert time from the format in which it is maintained by the system (that is, double precision fixed point binary number of micro seconds that have elapsed since 3/1/1900) into various EBCDIC forms of time and/or date. The privilege is the same as that of the calling routine, implying that SYSTIME is a fence straddler. (See Chart HF.)

Entry Point: SYSKA1 via type 1 linkage.

Input: General register 1 contains a pointer to the following parameter list:

Word 1 Address of a halfword containing the length in bytes of the text field. Maximum allowable length is 50 bytes.

Word 2 Address of the text field in which the output of SYSTIME is to be placed.

On input, the text field contains special character groups which specify to SYSTIME the form that its output should have. The special character groups are as follows:

YY	Year, from 00 to 99.
YYY	Year, from 1900 to 1999.
MO	Numeric month, from 01 to 12.
DDD	Day of year, from 001 to 366.
DD	Day of month, from 01 to 31.
HH	Hour of day, from 00 to 23.
MM	Minutes past hour, from 00 to 59.
SS	Seconds, from 00 to 59.
SSS	Tenths of seconds, from 000 to 599.
SSSS	Hundredths of seconds, from 0000 to 5999.

MON First 3 characters of month.
DAY First 3 characters of day.
DAYW First 4 characters of day.

Word 3 Zero, or the address of a binary number to be converted to time and/or date.

Output: All special character groups within the text field are converted. All characters which are not part of the special character group are unchanged. Register 15 contains the following information:

Bits 0-15 yy Year, from 00 to 99 in binary.

Bits 16-31 ddd Day of year, from 001 to 366 in binary.

Example: An input text field appearing,

'MO/DD/YYbHH:MMbHOURS'

would give the following output from SYSTIME:

'01/06/68b23:59bHOURS'

b = blank.

If the length field is zero (the field pointed to by the first word of the parameter list) or if the text field is zero or blank, a default field as follows is inserted in the text field, left justified:

'MO/DD/YYbHH:MM'

Module Called: REDTIME (CEAR6) Read elapsed time SVC. SVC 218.

Exits:

Normal - Return to calling routine.

Error - ERR1 ABEND Using register notation with length greater than 50.

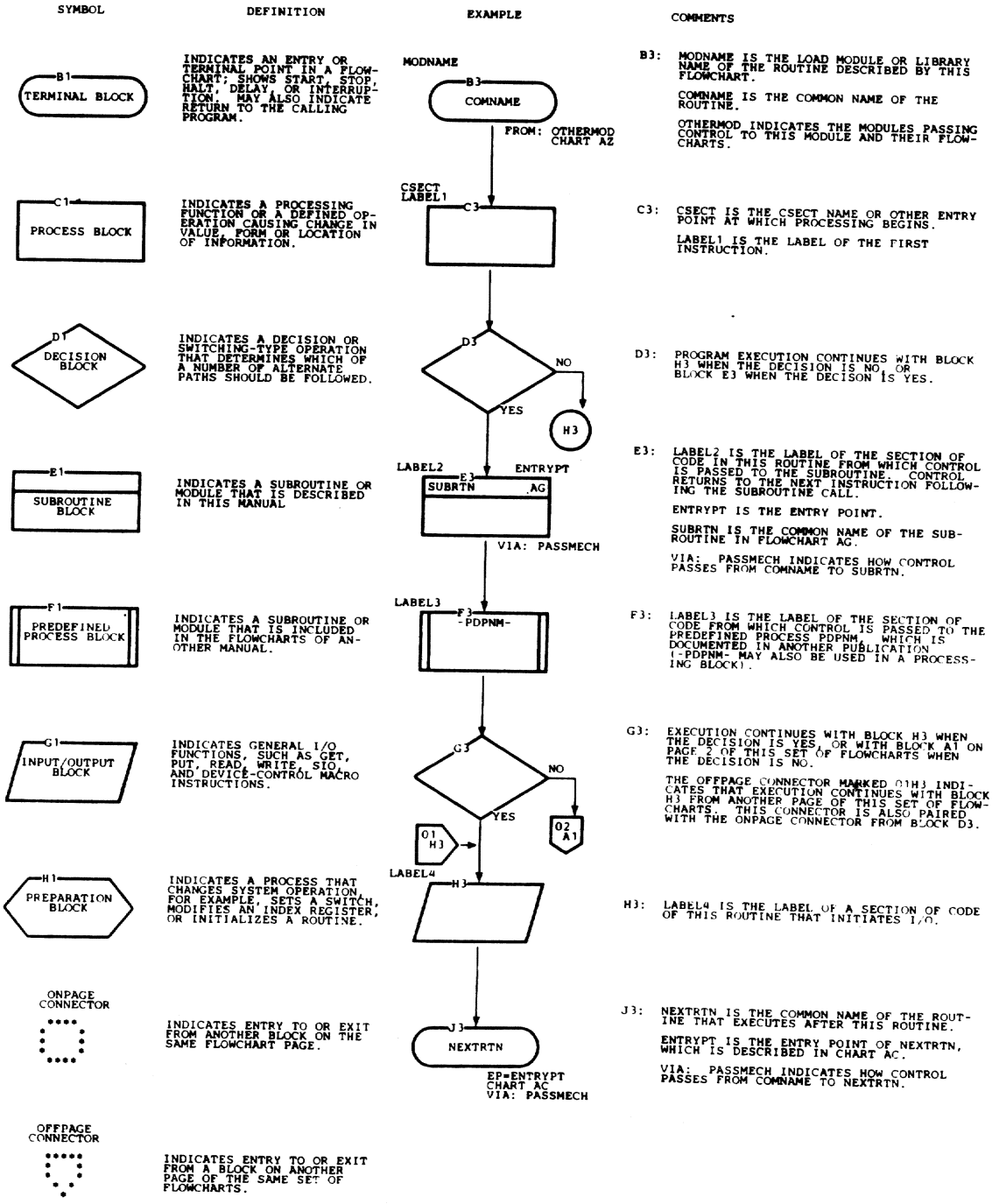
ERR2 ABEND Branch Array R2 value greater than 48 (i.e., indexing by R2 value would exceed length of table).

Operation: The date and time are computed from the time parameter supplied by the user or obtained from the REDTIME macro and are converted to the equivalent EBCDIC values.

The input text length is tested for zero. If it is zero, the default format of time and date is assumed. If the length is non-zero, the input field is scanned for special character groups. Each time a special character group is found, the appropriate date or time EBCDIC value is stored as indicated in the text. When the entire text has been processed, a return is made to the calling program.

SECTION 9: FLOWCHARTS

The flowcharts in this manual have been produced by an IBM program, using ANSI symbols. The symbols are defined in the left column below, and examples of their use are shown at the right.



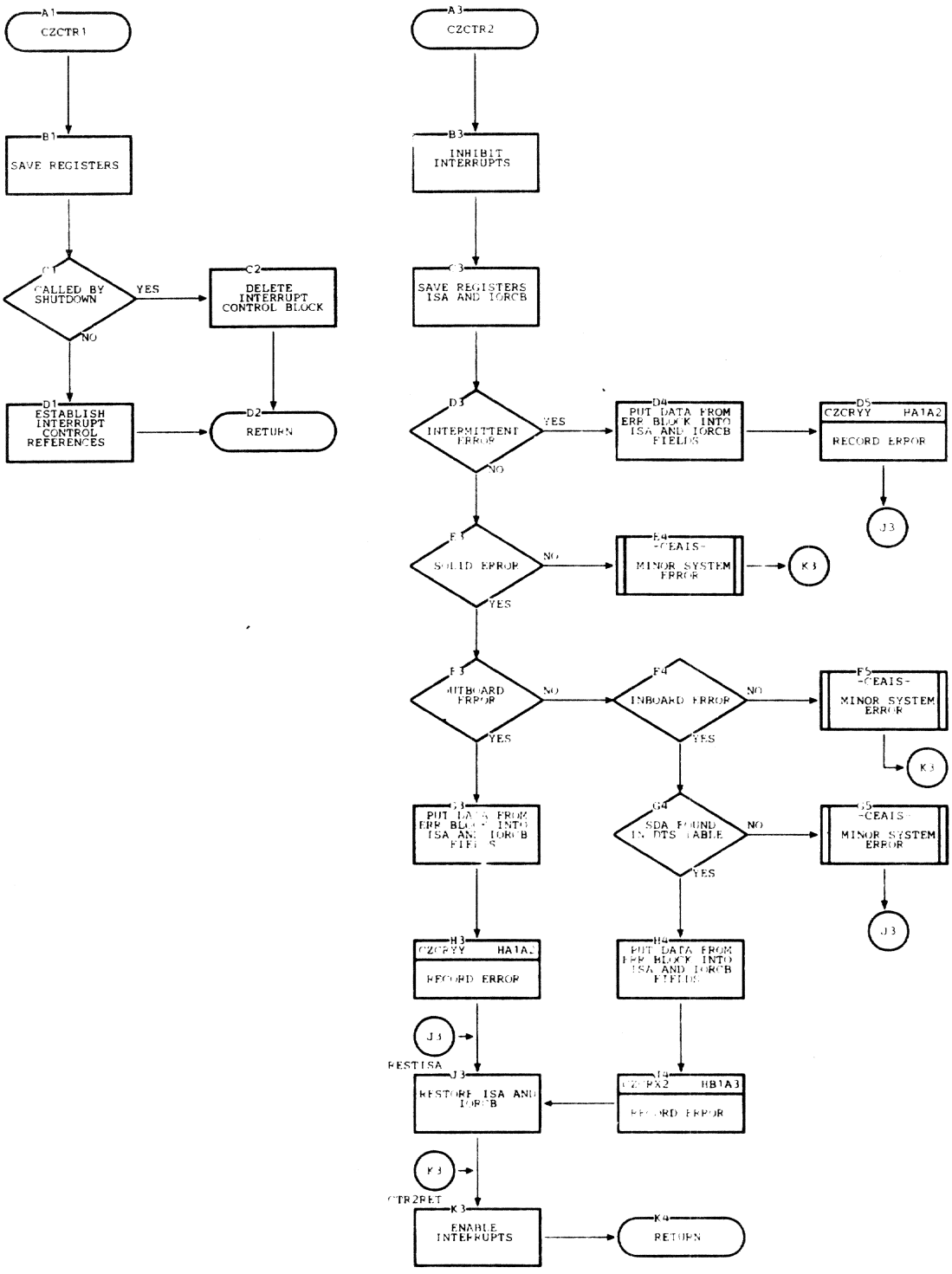
Program Logic Manual

GY28-2018-3

System Service Routines

Flowcharts on pages 95-258 were not scanned.

Chart HG. RERIM (CZCTR)



APPENDIX A: DATA SET CONTROL BLOCK (DSCB) FORMAT

Each SAM direct access volume has a volume table of contents (VTOC) that describes its contents; the VTOC contains all the data set control blocks (DSCBs) for the data sets contained on that volume.

Each VAM volume has a PAT table describing whether a page is used for DSCB or Data. The VAM DSCBs are of E and F formats.

For SAM formatted volumes, the TSS/360 DSCB formats are:

Format-0 - an unused DSCB; contains binary zeros.

Format-1 - the data set label for direct access volumes containing physical sequential data sets. (See Figure 9.)

Format-3 - used to describe additional extents of a physical sequential data set if there are more than can be described in a format-1 DSCB. (See Figure 10.)

Format-4 - the first DSCB in the VTOC. (See Figure 11.)

Format-5 - describes available extents on a volume containing physical sequential data sets. (See Figure 12.)

Format-A - the data set label for direct access volumes containing virtual storage data sets. (See Figure 13.)

Format-B - used to describe additional extents of a virtual storage data set if there are more than can be described in a format-A DSCB. (See Figure 14.)

Format-C - describes available extents on a volume containing virtual storage data sets. (See Figure 15.)

For VAM formatted volumes, the TSS/360 DSCB formats are:

Format-E - This DSCB is used for single or multi-volume VAM public or private data sets. (See Figure 16.)

Format-F - This DSCB is used as an extension of the format E DSCB. (See Figure 17.)

Field Number	Field Name	Identification	Length	Representation	Usage
1	Data Set Name	DSCNME	44 bytes	alphameric (EBCDIC)	May have one of two forms: 1. User ID.User Name 2. User ID.User Name.Generation The user ID is eight bytes in length. The user name is a maximum of 35 bytes in form 1, and 26 bytes in form 2. The generator is of the form GNNNNVNN (eight characters) where N is a decimal number. Note: If a user declares himself an OS/360 user, the User ID is not concatenated and he is allowed a 44-character name in form 1 and 35 characters in form 2.
2	Format Identifier	DSCFID	1 byte	hexadecimal	Contains X'F1'
3	Data Set Serial Number	DSCVSR	6 bytes	alphameric (EBCDIC)	Used for data/volume relationship. This field contains the volume serial number of the first (or only) volume which contains the data set.
4	Volume Sequence Number	DSCVSQ	2 bytes	binary	Used to indicate the order of the present volume relative to the first volume, containing the data set. Range is from 0001 to 9999.
5	Creation Date	DSCCRD	3 bytes	discontinuous binary	YDD where Y=year (0-99) and DD=day (1-366)
6	Expiration Date	DSCEXP	3 bytes	discontinuous binary	Indicates the year and day the data set may be purged. Has same form as the Creation Date.
7a	Number of Extents on Volume	DCBNEX	1 byte	binary	The number of total separate extents in which the data set resides on this volume; count does not include extent describing a user's label track.
7b	Number of Bytes Used in Last Directory Block	DCBFL1	1 byte	binary	Used only for an OS/360 partitioned data set. It contains a binary number indicating the total number of bytes being used in the last available directory block. Value of zero indicates that the last available block is not being used.
7c	Spare	DCBSP1	1 byte		
8	System Code	DCBSCD	13 bytes	alphameric (EBCDIC)	To identify the programming system. Only characters A-Z, 0-9, and blanks are used.
9	Reserved For Future Use		7 bytes		
10	File Type	DCBFTY	2 bytes	hexadecimal	Hex Code — Description 4000 — Sequential Organization 2000 — Direct Organization 0200 — Partitioned Organization 0000 — Organization not defined 8000 — Index Sequential Organization Note: If bit 7 of byte zero is on, the data set described by this DSCB must remain in the same absolute location of the direct access device.

Figure 9. Format-1 DSCB (Part 1 of 4)

Field Number	Field Name	Identification	Length	Representation	Usage
11	Record Format	DSCRFM	1 byte	binary	Bit Binary Code 0,1 10 = fixed-length record (F) 01 = variable-length record (V) 11 = undefined-format record (U) 2 = track overflow feature must be used 3 = blocked (B) 4 = truncated records in data set (T) 5,6 10 = control character ASA code (A) 01 = control character machine code (M) 00 = control character stated 7 Not used
12	Option Codes	DSCOPT	1 byte	binary	The 8 bits of this field are used to indicate various options used in building the data set. The bit configuration of this field is exactly the same as that described for the OPTCD field in the DCB. Bit 0 of this field is common across the various data set organizations, as follows: Bit Description 0 If on, indicates data set was created using Write Validity Check 1-7 reserved
13	Block Length	DSCBKS	2 bytes	binary	Block length for fixed-length records or maximum block size for variable-length records (1-32,767 bytes).
14	Record Length	DSCLRC	2 bytes	binary	Record length for fixed-length records or the maximum record length for variable-length records. (1-32,767 bytes)
15	Key Length	DSCKLN	1 byte	binary	Length (1-255 bytes) of the key of the data record in the data set. A value of zero means no key exists.
16	Key Location	DSCRKP	2 bytes	binary	High order position (byte 1 through byte 32,767) of the key in the data record. A value of zero indicates that the key is not in the data portion but corresponds to the physical key on the direct access volume.
17	Data Set Indicators	DSCDS1	1 byte	binary	Bit Description 0 If on, indicates that this is the last volume which normally contains this data set. 1 Spare 2 If on, indicates that the block length must always be a multiple of eight bytes. 3 If on, indicates that this data set is security protected and a password must be provided in order to access it. 4 (Data Set Abnormal Close) if on, indicates that this data set may be invalid due to abnormal termination of a task which was writing or updating the data set. 5-7 Spare

Figure 9. Format-1 DSCB (Part 2 of 4)

Field Number	Field Name	Identification	Length	Representation	Usage
18a	Original Request For Space		1 byte	binary	This field indicates the type of request that was issued for the initial allocation: Bit Description 0,1 00 indicates original request was in absolute tracks; no secondary allocation is made. 01 indicates request is in number of records. 10 indicates request is in number of tracks. 11 indicates request is in number of cylinders. 2 If on, indicates original request was for an indexed sequential data set with embedded index. 3 Spare 4 If on, indicates that the original request was for the maximum contiguous quantity on the volume. 5 If on, indicates that the original request was for the maximum contiguous quantity on the volume. 6 If on, indicates that the original request was for the five or fewer extents that were larger than or equal to a specified minimum. 7 If on, indicates that the original request was a record request and was to be rounded upward to a cylinder boundary.
18b	Secondary Allocation	DSCSAL	3 bytes	binary	The three bytes of this field are a binary number indicating how many blocks, tracks, or cylinders are to be requested at the end of the initial extent when processing a sequential data set.
19	Last Record Pointer	DSCLRD	5 bytes	binary	(Direct Access Address) -- identifies the last record. It is in the format TTRLL (where TT is the relative address of the track containing the last record, R is the ID on that track, and LL is the number of bytes remaining on that track following the record). If all five bytes equal binary zeros, the last record pointer does not apply.
20	Spare	DSCSP2	2 bytes		

Figure 9. Format-1 DSCB (Part 3 of 4)

Field Number	Field Name	Identification	Length	Representation	Usage
21	Extent Type Indicator Hex	DSCXTS	1 byte	hexadecimal	Hex Code / Meaning 00 — Next three fields do not indicate any extent. 01 — Prime Areas (Index Sequential) or consecutive area, that is, the extent containing the data records (user's). 02 — Overflow area of an indexed sequential data set. 04 — INDEX area of an Index Sequential data set. 40 — Next three fields indicate one track is used to contain user's data set labels. 80 — The extent described is sharing one or more cylinders with one or more data sets. 81 — The extent described begins and ends on cylinder boundaries, i.e., the extent is composed of one or more cylinders.
22	Extent Sequence Number	DSCMVL	1 byte	binary	This field uniquely identifies each separate extent on a given volume for a data set. For all organizations but indexed sequential the first extent of the data set on each volume is identified with a zero in this field. The first extent on each volume of an indexed sequential data set is identified with a value of one in this field. Additional extents on the volume are identified with sequentially increasing binary values. This field is always zero for an extent field pointing to a user label track.
23	Lower Limit	DSCLCH	4 bytes		(Direct Access Address) - the cylinder and track address (CCHH) specifying the starting point of this extent component.
24	Upper Limit	DSCLCH	4 bytes		(Direct Access Address) - the cylinder and track address (CCHH) specifying the ending point of this extent component.
25-28	Additional Extents	DSCEX2	10 bytes		Same as fields 21-24.
29-32	Additional Extents	DSCEX3	10 bytes		Same as fields 21-24.
33	Pointer to Next DSCB Record	DSCCN3	5 bytes		This field contains the CCHHR of a continuation DSCB, if needed, to further describe the data set. The next DSCB will be of the format-3 type. All zeros signify that this is the last DSCB. (If field 10 indicates Index Sequential Organization, this field will point to a format-2 DSCB.)

Figure 9. Format-1 DSCB (Part 4 of 4)

Field Number	Field Name	Identification	Length	Representation	Usage
1	Key Identification (in key)	DSVNME	4 bytes	hexadecimal	'03030303'
2-17	Extents (in key)	DSCEX4	40 bytes		Same as fields 21-24 of format-1 DSCB. Four extents.
18	Format Identifier	DSCFID	1 byte	hexadecimal	'F3'
19-54	Additional Extents	DSCEX9	90 bytes		Same as fields 21-24 of format-1 DSCB. Nine extents.
55	Pointer to Next DSCB	DSCCHN	5 bytes		CCHHR for next format-3 DSCB. Zero in field signifies this is the last DSCB.

Figure 10. Format-3 DSCB

Field Number	Field Name	Identification	Length	Representation	Usage
1	Key Field	VTC Key	44 bytes	hexadecimal	"04's"
2	Format ID	VTCFID	1 byte	hexadecimal	'F4'
3	Highest Prime CCHHR	VTCHPC	5 bytes		This field contains the address of the last active format-1 or -A DSCB. It is used to stop a search for a data set name.
4	Available DSCB Records	VTCHCT	2 bytes	binary	Count of number of unused records in VTOC.
5	The CCHH of the Next Available Alternate Track	VTCHCA	4 bytes		(See Note 1 in field 7.)
6	Number of Alternate Tracks	VTCNAT	2 bytes	binary	This field contains the number of alternate tracks available. For the 2321 the assigned alternate area will be the last four strips in each cell. (See Note 1 in field 7.)
7	VTOC Indicators	VTCFL1	1 byte		Bit 0 - If on, means that either no format-5 or -C DSCBs exist or that they do not reflect the true status of the volume. Bit 1 - If on, means that the format-C DSCB is being used for storage management. The volume is formatted for VAM. Bit 2 - If on, indicates that the volume is a system (public) volume, and should have space allocated (by TSS/360) accordingly. Bits 3-7 - Spare. Note 1: Alternate tracks will be assigned in ascending sequences. Thus, field 5 will be incremented and field 6 decremented by one when an alternate track is used.
8a	Number of Extents	VTCNEX	1 byte	hexadecimal	This field contains the hexadecimal constant '01' to indicate one extent in the VTOC.

Figure 11. Format-4 DSCB (Part 1 of 3)

Field Number	Field Name	Identification	Length	Representation	Usage
8b	Reserved for Future Use	VTC$02	2 bytes		
9	Device Constants	VTCDVC	14 bytes	binary	This field contains physical constants describing the device on which this volume was mounted when the VTOC was created. This field contains the following information: Device Size (2 bytes) - The number of logical cylinders in a volume on this device. (A logical cylinder is the smallest collection of two or more tracks that can be protected by a Set File Mask CCW.) Logical Cylinder Size (2 bytes) - The number of tracks in a logical cylinder on this device. Track Length (2 bytes) - The number of available bytes on a track exclusive of home address and record zero (record zero is assumed to be a non-keyed record with an eight byte data field). Record Overhead (3 bytes) - The number of bytes required for gaps, check bits, and count field for each record. This value varies according to the record characteristics and thus is broken down into three subfields: I Overhead required for a keyed record other than the last record on the track. This is the first byte. L Overhead required for a keyed record that is the last record on the track. This is the second byte. K Overhead bytes to be subtracted from the I and L bytes if the record does not have a key field. This is the third byte. Flag (1 byte) - Further defines unique characteristics of the device. Bit Description 0-4 Reserved for future use. 5 CCHH of an absolute address (CCHH) is used as a continuous binary value as in the case of the 2301. 6 CCHH of an absolute address (CCHH) is used as four separate one-byte binary values as in the case of the 2321. 7 A tolerance factor must be applied to all but the last record on the track. Note that if bits 5 and 6 are off, the CC and the HH of an absolute address (CCHHR) are used as half-word binary values as in the case of the 2311.

Figure 11. Format-4 DSCB (Part 2 of 3)

Field Number	Field Name	Identification	Length	Representation	Usage
					Tolerance (2 bytes) - A value that when divided by 512 is used to determine the effective length of the record on the track. DSCBs per track (1 byte) - Contains the number of full records of 44-byte key and 96-byte data length that can be contained on one track of this device. Directory blocks per track (1 byte) - Contains the number of full records of 8-byte key and 256-byte data length that can be contained on one track of this device.
10A	Spare	VTC$04	19 bytes		
10B	Gross Available Space		5 bytes	hexadecimal	Right-most four bytes indicate number of pages available if it is a VAM volume (field 7, bit 1 on); otherwise the first two bytes indicate the number of cylinders which are entirely free and the next two bytes indicate the tracks not on the cyclinder which are free. Left-most byte (X'FF') indicates space allocation by TSS/360 only.
10C	Pointer to Format-6 DSCB	VTCPTR	5 bytes	binary	This field contains the CCHHR of the first format-6 DSCB if it exists. Otherwise it contains binary zeros.
11-14	VTOC Extent		10 bytes		These fields describe the extent of the VTOC, and are identical in format to fields 21-24 of the format-1 DSCB.
15	Spare	VTC$05	25 bytes		

Figure 11. Format-4 DSCB (Part 3 of 3)

Field Number	Field Name	Identification	Length	Representation	Usage
1	Key Identification (in Key)	DASKEY	4 bytes	hexadecimal	'05050505'
2	Available Extent (in key)	DASE1	5 bytes		Extent of space available for allocation to a data set. First two bytes = relative track address. Next two bytes = number of full cylinders included in extent. The last byte = number of tracks in addition to cylinders in the extent.
3-9	Available Extents (in Key)	DASE2	35 bytes		Same as field 2; they are in relative track address sequence.
10	Format Identifier	DASFID	1 byte	hexadecimal	'F5'
11-28	Available Extents	DASE9	90 bytes		Same as field 2. There are 26 available extent fields in this DSCB (key and data).
29	Pointer to Next Format 5	DASCHA	5 bytes	binary	This field contains the CCHHR address of the next format-5 DSCB if it exists. It contains binary zeros otherwise.

Figure 12. Format-5 DSCB

Field Number	Field Name	Identification	Length	Representation	Usage
1	Data Set Name	DSVNME	44 bytes	alphameric (EBCDIC)	Same as format-1, field 1.
2	Format Identifier	DSVFID	1 byte	hexadecimal	'FA'
3a	Date Last Used	DSVDLU	3 bytes		YDD, where Y=year (0-99) and DD=day (1-366).
3b	Rate of Usage	DSVROU	3 bytes		
4	Volume Sequence Number	DSVVSQ	2 bytes	binary	Same as format-1, field 4.
5	Creation Date	DSVCRD	3 bytes	discontinuous binary	Same as format-1, field 5.
6	Expiration Date	DSVEXD	3 bytes	discontinuous binary	Same as format-1, field 5.
7a	Number of Extents of Volume	DSVNEX	1 byte		
7b	Number of Bytes Used in Last Data Page	DSVLPB	2 bytes	binary	Same as format-1, field 7b.
8	System Code	DSVSCD	13 bytes	alphameric (EBCDIC)	Same as format-1, field 8.
8a	PAD for index sequential data set	DSVXPD	1 byte		
9	Spare	DSV 01	6 bytes		
10	File Type	DSVFTP	2 bytes		X'7100' VAM Index Sequential X'7200' VAM Sequential X'7300' VAM Partitioned Index Sequential X'7400' VAM Partitioned Sequential X'7500' VAM Partitioned
11	Record Format	DSVRFM	1 byte	binary	Same as format-1, field 11.
12	Option Codes	DSVOPC	1 byte	binary	Same as format-1, field 12.
13	Record Length	DSVRCL	4 bytes	binary	Same as format-1, field 14.
14	Key Length	DSVKYL	1 byte	binary	Same as format-1, field 15.
15	Key Location	DSVKLC	2 bytes	binary	Same as format-1, field 16.
16	Data Set Indicators	DSVDSI	1 byte	binary	Same as format-1, field 17.
17a	Original Request for Space Indicators		1 byte	hexadecimal	X'00' No secondary allocation.
17b	Secondary Allocation	DSVSAL	3 bytes	binary	Same as format-1, field 18.

Figure 13. Format-A DSCB (Part 1 of 2)

Field Number	Field Name	Identification	Length	Representation	Usage
18	Number of Data Pages	DSVNDP	2 bytes		
19	Number of Directory Pages	DSVDOP	2 bytes		Number of page in POD or index directory.
19a	Number of Overflow Pages	DSVNOP	1 byte		
20	Total Number of Pages Thus Far Assigned	DSVTNP	2 bytes		
21	Extent		6 bytes		
21a 21b 21c 21d		DSVXTS			Bits Description 2 00 = pages are in use 01 = pages are not in use 14 number of continuous external pages 16 first external page number 16 first virtual page number
22-25	Extents		24 bytes		Four more extents.
26	Next DSCB	DSVWXT	5 bytes	binary	CCHHR of a continuation DSCB (format-B). If there are no format-B DSCBs for the data set, then this field contains binary zeros.

Figure 13. Format-A DSCB (Part 2 of 2)

Field Number	Field Name	Identification	Length	Representation	Usage
1	Key Field (in key)	DSV7KY	2 bytes	hexadecimal	'0B0B'
2	Extent (in key)	DSV7X1	42 bytes		Seven extents as described in format-A, field 21.
9	Format Identifier	DSV7ID	1 byte	hexadecimal	'FB'
10-24	Extents	DSV7XZ	90 bytes		15 extents as described in format-A, field 21.
25	Pointer to Next format-B DSCB	DSV7NX	5 bytes	binary	CCHHR format. Binary zero if there are no chained format-B DSCBs.

Figure 14. Format-B DSCB

Field Number	Field Name	Identification	Length	Representation	Usage
1	Key Field (in key)	DAVKEY	4 bytes	hexadecimal	'0C0C0C0C'
2	Extents (in key)	DAVE1	4 bytes		
2a		DAVE11			Bits Description 16 number of continuous external pages
2b		DAVE12			16 first external page number
3-11	Extents (in key)	DAVE2	36 bytes		Same as field 2 above.
12	Format Identifier	DAVFID	1 byte	hexadecimal	'FC'
13	Spare	DAVS1	3 bytes		
14-34	Extents	DAVEB DAVE1F (last extent)	84 bytes		Same as field 2 above.
35	Spare	DAVS2	3 bytes		
36	Pointer to Next Format-C (DSCB)	DAVCHA	5 bytes	binary	CCHHR format. Binary zero if there are no chained format-C DSCBs.

Figure 15. Format-C DSCB

Byte No.	ID	Full Name	Length (bytes)	Representation
00	DSENME	Data Set Name	44	
44	DSESCD	System Code	13	
57	DSEXPD	Pad for Indexed Sequential	1	
58	DSELPB	No. of Bytes in Last Data Page	2	
60	DSERFM	Record Format	1	
61	DSEOPC	Option Codes	1	
62	DSEFTP	File Type	2	
64			1	Spare
65	DSERCL	Record Length	3	
68	DSEDSI	Data Set Indicators	1	
69	DSEKYL	Key Length	1	
70	DSEKLC	Key Location	2	
72	DSESAI	Secondary Allocation Indicator	1	
73	DSESAL	Secondary Allocation	3	

Figure 16. Format-E DSCB (Part 1 of 2)

Byte No.	ID	Full Name	Length (bytes)	Representation
76	DSENDP	No. of Data Pages	2	
78	DSEDOP	No. of Directory Pages	2	
80	DSENOP	No. of Overflow Pages	1	
81	DSENVL	No. of Private Volumes	1	
82	DSETNP	Data Set Size at Close	2	
84			1	Spare
85	DSECRD	Reference Date	3	
88	DSEEXD	Change Date	3	
91			5	Spare
96				List of Vol. IDs for volumes of a private data set in 6-byte entries. Length is dependent on the value of DSENVL
	DSEENT	Page Entries (1 word)		Format: AF(2) | (2) | Rel Vol No(12) | Ext Pg. No(16)
248	DSECHN	Pointer to Next DSCB	4	
252	DSETYP	DSCB Type	1	
253			1	Spare
254	DSECKS	Checksum	2	

Figure 16. Format-E DSCB (Part 2 of 2)

Byte No.	ID	Field Name	Length (bytes)	Representation
0				List of volume IDs for volumes of a private data set in 6-byte entries. Variable length field depending on value of DSENVL from the format-E DSCB.
	DSFENT	Page Entries		Format: AF(2) | | Rel Vol No(12) | Ext Pg. No(16)
248		Pointer to next DSCB	4	Format: DSCB Slot (4) | | Rel Vol No(12) | Ext Pg. No(16)
252	DSFTYP	DSCB Type	1	
253			1	Spare
254	DSFCKS	Checksum	2	

Figure 17. Format-F DSCB

APPENDIX B: CATALOG SBLOCK FORMAT

Each member in the catalog data set is comprised of an integral number of pages. Each page is divided into 64-byte blocks, called SBLOCKs, which serve as the basic unit of storage within the catalog.

Each logical entity within the catalog is comprised of a chain of one or more SBLOCKs. These logical entities are:

- Indexes
- Generation Indexes
- Data Set Descriptors
- Sharing Descriptors
- Sharer Lists

Catalog service routines receive data in varied parameter lists and pack it into available SBLOCKs. Data is retrieved from the catalog, via the LOCATE routine, in the SBLOCK form. Various command language service routines retrieve data in this form.

SBLOCK Format

The first eight bytes of an SBLOCK always have the same format. The last 56 bytes of an SBLOCK vary in format depending on the logical entity which the SBLOCK is part of.

Figure 18 shows the format of a generalized SBLOCK; Figures 19 through 25 illustrate specific SBLOCK formats.

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
1	1-3	3	Forward pointer to the first character of the next SBLOCK in the chain. Pointer is of the form Pbb where P is the logical page number within the member and bb is the relative byte within the page.	CCCFWD
2	4	1	Binary count of SBLOCKs allocated from a page. This field is maintained by Catalog Services in the first SBLOCK of each page.	CCCCT1
3	5-7	3	Backward pointer to the preceding SBLOCK in a chain. Pointer is of the form Pbb.	CCCBWD
4	8	1	Binary count of bytes allocated from the field to follow.	CCCCT2
5	9-64	56	Allocatable field; format is variable according to SBLOCK usage. See following descriptions for this field.	

Figure 18. General SBLOCK Format

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK control field (see Table 11)	
1	9-16	8	Name of Data Set Descriptor; left adjusted and padded with blanks.	CCCNME
2	17	1	Identification flag: X '03' Data Set Descriptor - Private DSD X '06' Data Set Descriptor - Public DSD	CCFL1
3	18-20	3	Pointer to Sharer List. Pointer is of the form Pbb, where: P = page number bb = location within the page	CCCPTL
4	21	1	Share flags; X'00' Private X'01' Shared Universally X'02' Shared by Listed Sharers	CCCFL2
5	22	1	Share privileges; used only if the Data Set Descriptor is universally shareable: X'00' Unlimited Access X'01' R/W Access X'02' RO Access	CCCFL3
6	23	1	RET Parameter Storage Type | Deletion | Owner Access 00 | 0000 | 00 Storage Type 00 - Permanent 10 - Temporary Deletion 0000 - No Deletion 0001 - Delete at LOGOFF 0010 - Delete at CLOSE Owner Access 01 - R/W Access 10 - RO Access	CCCFL4
7	24	1	Label Data: X'01' No labels (tape only) X'02' Standard labels X'04' Standard and user labels	CCCLAB
8	25	1	Data set organization	CCCORG
9	26-40	15	User data; includes accounting and frequency of use data which is maintained by the Command System.	CCCUSE
10	41-44	4	DSCB Pointer DSCB No(4) | Rel Vol No(12) | Page No(16)	
11	45-50	6	Spare	

Figure 19. SBLOCK Format - Data Set Descriptor (First Block) (Part 1 of 2)

Field	Bytes	Length (Bytes)	Description	Symbolic ID
12	51-64	14	Volume Information	
			51 52 53 56 57 60 61 64	
			Public: Spare | Type | Spare | Reserved	
			Private SAM: Vol Count | Type | Vol ID | File Seq. No.	
			Private VAM: Spare | Type | Vol ID | Spare	

Figure 19. SBLOCK Format - Data Set Descriptor (First Block) (Part 2 of 2)

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK Control field (see Table 11)	
1	9-20	12	Volume field: bytes 0- code identifying type of underlying device bytes 4-9 volume serial number bytes 10-11 file sequence number	
2	21-32	12	volume field - same as field 1	
3	33-94	12	volume field - same as field 1	
4	45-56	12	volume field - same as field 1	
5	57-64	8	Unused	

Figure 20. SBLOCK Format - Data Set Descriptor (Chained SBLOCKs)

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK control field (see Table 11)	
1	9-16	8	Name of Index; left adjusted and padded with blanks.	CCCNME
2	17	1	Identification Flags: X'01' Index X'02' Generation Index	CCCFL1
3	18-20	3	Pointer to Sharer List. Pointer is of the form Pbb, where: P = page number bb = location within the page	CCCPTL
4	21	1	Share Flags: X'00' Private X'01' Shared Universally X'02' Shared by Listed Sharers	CCCFL2
5	22	1	Share privileges; used only if index is universally sharable: X'00' Unlimited Access X'01' R/W Access X'02' RO Access	CCCFL3
6	23-24	2	Binary count of pointers in the index.	CCCCT4
7	25-26	2	Binary count of maximum number of generations allowed.	CCCCT9
8	27	1	Generation Flags: bits 0-3 X'1' Save deleted generations X'2' Scratch deleted generations bits 4-7 X'1' Delete oldest generations X'2' Delete all generations in index	CCCFL5
9	28	1	Interlock byte	CCCILK
10	29-40	12	Pointer Entry: bytes 0-7 Name of entity pointed to; left adjusted and padded with blanks byte 8 Flags to indentify entity pointed to: X'01' Index X'02' Generation Index X'03' Data Set Descriptor X'04' Sharing Descriptor bytes 9-11 Pointer of the form Pbb, where: P = page number bb = location within the page	
11	41-52	12	Pointer entry - same format as field 10.	
12	53-64	12	Pointer entry - same format as field 10.	

Figure 21. SBLOCK Format - Index (Generation Index) -- First SBLOCK

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK control field (see Table 11)	
1	9-20	12	Pointer entry: bytes 0-7 Name of entity pointed to; left adjusted and padded with blanks byte 8 Flags to identify entity pointed to: X'01' Index X'02' Generation Index bytes 9-11 Pointer of the form Pbb, where: P = page number bb = location within the page	
2	21-32	12	Pointer entry - same as field 1	
3	33-44	12	Pointer entry - same as field 1	
4	45-56	12	Pointer entry - same as field 1	
5	57-64	8	Unused	

Figure 22. SBLOCK Format - Index (Generation Index) - Chained SBLOCK

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK control field (see Table 11)	
1	9-16	8	Name of Sharing Descriptor; left adjusted and padded with blanks.	CCCNME
2	17	1	Identification flag: X'04' sharing descriptor	CCCFL1
3	18-61	44	Owner's name for shared catalog level; left adjusted and padded with blanks.	CCCNM0
4	62-64	3	Unused	

Figure 23. SBLOCK Format - Sharing Descriptor

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK control field (see Table 11)	
1	9-10	2	Binary count of number of shares	CCCCT5
2	11-16	6	Unused	
3	17	1	Identification flag: X'05' Share List	CCCFL1
4	18-26	9	Sharer entry: bytes 0-7 Share indentification (3-8 characters); left-adjusted and padded with blanks byte 8 Share privileges: X'00' Unlimited access X'01' R/W Access X'02' RO Access	CCCNM2 CCCFL8
5	27-35	9	Sharer entry - same format as field 4	
6	36-44	9	Sharer entry - same format as field 4	
7	45-53	9	Sharer entry - same format as field 4	
8	54-62	9	Sharer entry - same format as field 4	
9	63-64	2	Unused	

Figure 24. SBLOCK Format - Sharer List (First SBLOCK)

Field #	Bytes	Length (Bytes)	Description	Symbolic ID
	1-8	8	SBLOCK control field (see Table 11)	
1		9	Sharer entry: bytes 0-7 Sharer indentification (3-8 characters); left-adjusted and padded with blanks. byte 8 Share privileges: X'00' Unlimited Access X'01' Read/Write Access X'02' Read only Access	
2	9-17	9	Sharer entry - same as field 1	
3	18-26	9	Sharer entry - same as field 1	
4	27-35	9	Sharer entry - same as field 1	
5	36-44	9	Sharer entry - same as field 1	
6	45-53	9	Sharer entry - same as field 1	
7	54-62	9	Sharer entry - same as field 1	
8	63-64	2	Unused	

Figure 25. SBLOCK Format - Sharer List (Chained SBLOCKs)

This module directory is presented as an aid to the field engineer. It provides a convenient cross-reference between the PLM and the microfiche, enabling the FE to quickly find the microfiche card that contains a particular entry point name, control section name, or module name.

Figure 26 is an alphabetical listing of the modules contained in this PLM, according to the module title (for example, OBTAIN). In addition to the title of the module, this table provides the module name (for example, CZCFO), the type of service routine to which the module belongs, the related flowchart, and a brief synopsis of the functions the module performs.

Figure 27 lists the modules alphabetically, according to module name (for example, CZCFO), and is intended for use when working with a dump containing load module names. The table identifies the module title (for example, OBTAIN), the flowchart, CSECT, and entry points. The module may then be found on microfiche, via the module name.

Title	Module Name	Type of Service Routine	Synopsis	Chart ID
ADDCAT	CZCFA	Catalog Services	1. Creates a data set descriptor in the user's catalog, associating the VTOCs of a data set with the index levels in the user's catalog. 2. Creates any index levels defined by the FQN which must precede the data set descriptors and do not currently exist. 3. Allows updating of a data set descriptor. 4. Controls the number of generations allowed under a generation index by performing deletion of out-moded generations, as required.	AA
ADDDSCB	CZCEK	External Storage Allocation (ESA)	Assign space for a new format-E or format-F DSCB.	BJ
ALLOCATE	CZCEA	ESA	Provides the initial allocation of direct access storage for new output data sets, one volume per allocation.	EA
BUMP	CZCAB	Device Management	Used to mount subsequent volumes of a multi-volume SAM data set.	CD
CATALOG ERROR PROCESSOR	CZCFE	Catalog Services	Invokes a completion code 1 ABEND whenever a catalog service routine encounters a user input data format error. Also, executes a SYSER and ABEND and writes a message to SYSLOG describing the type and location of the error in the catalog.	AM
CATFLUSH	CZCFX	Catalog Services	Copies members of the scratch catalog to individual user catalogs at task termination.	AL
CSECT STORE	CZCKZ	CSECT Store	Processes user requests made through the CSTORE macro, which enables the user to create, during program execution, a control section that is placed in the current job library as a module.	GA
DELCAT	CZCFD	Catalog Services	1. Deletes index levels from the catalog structure. 2. Recatalogs index levels under a different fully qualified name (FQN).	AB

Figure 26. Module Directory, Indexed Alphabetically by Module Title (Part 1 of 4)

Title	Module Name	Type of Service Routine	Synopsis	Chart ID
Drum Access Module (DRAM)	CZASY	Serviceability Aids	Used by those virtual memory programs, such as VMER and VMEREP, which must access the error records stored on the dummy spaces of the paging drum.	HC
DSCB/CAT RECOVERY	CZUFX	Catalog Services	Rebuilds a user catalog if the current member in the scratch catalog is unusable; rebuilds a member in the scratch catalog if the user catalog is unusable.	AK
DSCBREC	CZCEF	ESA	Used to recover from a checksum error, if possible.	BM
EREP67	CMASN	Serviceability Aids	Stand alone program, used to edit and print the information recorded on the paging drum by error recording programs.	HE
ESA LOCK	CZCEJ	ESA	Used to set, clear and record virtual memory locks.	BQ
EXTEND	CZCEX	ESA	Secondary allocation routine, called when additional space on a direct access volume is required for a data set.	BD
FINDEXPG	CZCEL	ESA	Gets pages of external storage for a data set.	BI
GET SBLOCK	CZCFG	Catalog Services	Receives a pointer containing the relative address of an SBLOCK and calculates its virtual storage address for the user.	AF
GIVBKSAM	CZCEG	ESA	Returns unused external storage from physical sequential data sets to ESA control, and deletes the references to the storage from the format -1 and -3 data set DSCBs. It is called only by SAM CLOSE.	BE
INDEX	CZCFI	Catalog Services	Constructs chained index levels in the catalog, and creates new members within the catalog data set.	AI
LOCATE	CZCFL	Catalog Services	Locates and returns SBLOCKs from the catalog, either by name or relative address.	AH
MERGE SAM	CZCEE	ESA	Returns extents from physical sequential data sets passed to it by scratch or GIVBKSAM, and merges them with the DADSM extents on their volume.	BF
MOUNTVOL	CZCAM	Device Management	Mounts, initializes, and builds PVT for all volumes of a VAM private data set.	CE
MTREQ	CZCAA	Device Management	Allocates and mounts unit record devices, such as disk or tape drives, used for private volumes. This routine can also be called by privileged system programs to obtain the symbolic device address of a public volume.	CA
OBTAIN/RETAIN	CZCFO	ESA	Reads VTOC and places DSCBs and volume labels in designated virtual storage locations. Writes DSCBs, user labels, and end-of-file markers to specified addresses.	BG
PAUSE	CZCAC	Device Management	Sends mount request messages to the system operator, asking him to mount volumes or to ready unit-record devices. It also verifies the operator's reply and, for tape or direct access, checks the label of the newly-mounted volume.	CB

Figure 26. Module Directory, Indexed Alphabetically by Module Title (Part 2 of 4)

Title	Module Name	Type of Service Routine	Synopsis	Chart ID
READWRIT	CZCEM	ESA	Reads DSCB pages into virtual storage and writes DSCB or PAT pages to external storage.	BP
RELEAS	CZCAD	Device Management	Informs the system that the device upon which a private volume was mounted is now free for other use, and notifies any task awaiting the freed device that it is now available.	CC
RELEXPG	CZCEN	ESA	Makes external and DSCB pages available to the system.	BI
RENAME	CZCFZ	Catalog Services	Changes the name in the key field of a format -1 or -A DSCB to the name specified for all volumes by the calling program.	BH
RERIM	CZCTR	Serviceability Aids	Passes error information between the RTAM subsystem and the subroutines VMER and VMSDR.	HG
SAM SEARCH	CZCEC	ESA	Called by either ALLOCATE or EXTEND to search the DADSM-DSCBs for available space to fill a request. The DADSM DSCBs are then updated to reflect the allocation.	BB
SCRATCH	CZCES	ESA	Deletes data set DSCBs on all volumes of a specified data set and assimilates the external storage back into the available space (the DADSM) on the volume.	BC
SEARCH SBLOCK	CZCFH	Catalog Services	Acquires and chains an empty SBLOCK as either an extended SBLOCK of a cataloged entity, or as the first SBLOCK of a cataloged entity.	AG
SHARE	CZCFS	Catalog Services	Adds sharing privileges to a catalog level. An unshared level can be set to sharable, or a shared level can have its sharing access modified.	AC
SHAREUP	CZCFU	Catalog Services	Links one user's private catalog to a level in another user's private catalog that is sharable.	AE
SVMA (Small Virtual Memory Allocation)	CZCHA	SVMA	SVMA serves the same function for bytes that VMA serves for pages. SVMA calls VMA for bytes totaling integral number of pages.	EA
SYSEARCH	CGCKC	Symbolic Library Service	Called by a system or user program to locate any individual parcel of a symbolic library, using its index as created by SYSINDEX.	FC
SYSINDEX	CGCKA	Symbolic Library Service	Automatically indexes the symbolic component of a symbolic library to create an alphabetical index of all the parcels, and is invoked when the user issues the appropriate RUN command or executes a program calling this routine.	FA
SYSTIME	CZCTA	Serviceability Aids	Converts time from system format to EBCDIC format.	HF
SYSXBLD	CGCKB	Symbolic Library Service	Builds the symbolic library index by scanning each line of the source data set to locate, extract, and place entries in a temporary index, and then forming a final and complete index.	FB
UNSHARE	CZCFV	Catalog Services	Removes sharing privileges from a catalog level.	AD

Figure 26. Module Directory, Indexed Alphabetically by Module Title (Part 3 of 4)

Title	Module Name	Type of Service Routine	Synopsis	Chart ID
USERCAT SCAN	CZUFY	Catalog Services	Rebuilds the SYSSVCT data set after OPEN VAM or DSCB/CAT RECOVERY discovers an error when reading the DSCB for SYSSVCT.	AJ
VAMINIT	CZCEQ	ESA	Initializes private VAM volumes when they are entered into the system.	BO
VMA (Virtual Memory Allocation)	CZCGA	VMA	A centralized routine which dynamically services all requests for virtual memory issued by the system or user's programs during the execution of a task. There are six entry points to VAM: GETMAIN (CZCGA2) - Get virtual storage by pages. FREEMAIN (CZCGA3) - Free virtual storage by pages. EXPAND (CZCGA4) - Expand an existing block of virtual memory. GETSMAIN (CZCGA6) - Get shared virtual storage. CONNECT (CZCGA7) - Connect to a shared page table. DISCONNECT (CZCGA8) - Disconnect from a shared page table.	DA and DB
VMER (Virtual Memory Error Recording)	CZCRX	Serviceability Aids	Informs the operator of a failing task I/O component, if the immediate report flag is on, and generates I/O error records that are to be output for preservation recording on drum via the Drum Access Module (DRAM).	HB
VOLSRCH	CZCEH	ESA	Determines the most suitable volume from which to allocate space.	BK
VMEREP	CZASE	Serviceability Aids	Retrieves, formats, and prints the environment recording information stored on the paging drum.	HD
VMSDR	CZCRY	Serviceability Aids	Accumulates error statistics on task I/O devices in the Statistical Data Table (SDT) and calls VMER to record I/O errors. It is called when a task I/O retry operation either ends successfully (intermittent outboard failure) or is completed with error after a prescribed number of retries (solid outboard failure).	HA
WRITDSCB	CZCEW	ESA	Used to construct a DSCB chain from the RESTBL, JFCB, DCB, and PVT.	BN

Figure 26. Module Directory, Indexed Alphabetically by Module Title (Part 4 of 4)

Module Name	Title	CSECT	Entry Point	Chart ID
CGCKA	SYSINDEX	CGCKAA	SYSINDEX	FA
CGCKB	SYSXBLD	CGCKBA	SYSXBLD	FB
CGCKC	SYSEARCH	CGCKCA	SYSEARCH	FC
CMASN	EREP67		Stand Alone	HE
CZASE	VMEREP	CZASEC	CZASE1	HD
CZASY	DRAM	CZASY	CZASY1	HC
CZCAA	MTREQ	CZCAAC	CZCAA1, CZCAA3	CA
CZCAB	BUMP	CZCABV	CZCAB1	CD
CZCAC	PAUSE	CZCACC	CZCAC1, CZCAC2	CB
CZCAD	RELEAS	CZCADU	CZCAD1, CZCAD2, CZCAD3	CC
CZCAM	MOUNTVOL	CZCAMC	CZCAM1	CE
CZCEA	ALLOCATE	CZCEAC	CZCEA1	BA
CZCEC	SAMSEARCH	CZCECC	CZCEC1	BB
CZCEE	MERGESAM	CZCEEC	CZCEE1	BF
CZCEF	DSCBREC	CZCEFC	CZCEF1	BM
CZCEG	GIVBKSAM	CZCEGC	CZCEG1	BE
CZCEH	VOLSRCH	CZCEHC	CZECH1	BK
CZCEJ	ESA LOCK	CZCEJC	CZCEJ1, CZCEJ2, CZCEJ3	BQ
CZCEK	ADDDSCB	CZCEKC	CZCEK1	BJ
CZCEL	FINDEXPG	CZCELC	CZCEL1	BI
CZCEM	READWRIT	CZCEMC	CZCEM1	BP
CZCEN	RELEXPG	CZCENC	CZCEN1	BL
CZCEQ	VAMINIT	CZCEQC	CZCEQ1	BO
CZCES	SCRATCH	CZCESC	CZCES1	BC
CZCEW	WRITDSCB	CZCEWC	CZCEW1	BN
CZCEX	EXTEND	CZCEXC	CZCEX1, CZCEX2	BD
CZCFA	ADDCAT	CZCKAT	CZCFA1, CZCFA2	AA
CZCFD	DELCAT	CZCFDY	CZCFD1	AB
CZCFE	CATALOG ERROR PROCESSOR	CZCFEC	CZCFE1	AM
CZCFG	GET SBLOCK	CZCFGY	CZCFG1, CZCFG2, CZCFG3 CZCFG4	AF
CZCFH	SEARCH SBLOCK	CZCFHC	CZCFH1	AG
CZCFI	INDEX	CZCIND	CZCFI1	AI
CZCFL	LOCATE	CZCFLY	CZCFL1	AH
CZCFO	OBTAIN/RETAIN	CZDFOB	CZCFO1, CZCFO2, CZCFR1	BG
CZCFS	SHARE	CZCFSC	CZCFS1	AC
CZCFU	SHAREUP	CZCFUC	CZCFU1	AE
CZCFV	UNSHARE	CZCFVC	CZCFV1	AD
CZCFX	CATFLUSH	CZCFXC	CZCFX1, CZCFX2, CZCFX3, CZCFX4, CZCFX5, CZCFX6, CZCFX7, CZCFX8	AL
CZCFZ	RENAME	CZCFZC	CZCFZ1	BH
CZCGA	VMA	CZCGAC	CZCGA2 - GETMAIN (pages) CZCGA3 - FREEMAIN (pages) CZCGA4 - EXPAND CZCGA6 - GETSMAIN CZCGA7 - CONNECT CZCGA8 - DISCONNECT	DA
CZCHA	SVMA	CZCHAC	CZCHA2 - GETMAIN (bytes) CZCHA3 - FREEMAIN (bytes)	EA
CZCKZ	CSECT STORE		CZCKZ1	GA
CZCRX	VMER	CZCRXC	CZCRX1, CZCRX2, CZCRX3 CZCRX4	HB
CZCRY	VMSDR	CZCRYC	CZCRYY	HA
CZCTA	SYSTIME		SYSKA1	HF
CZCTR	RERIM	CZCTRC	CZCTR1, CZCTR2	HG
CZUFX	DSCB/CAT RECOVERY	CZUFXC	CZUFX1, CZUFX2	AK
CZUFY	USERCAT SCAN	CZUFYC	CZUFY1, CZUFY2	AJ

Figure 27. Module Directory, Indexed Alphabetically by Module Name

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

IBM System/360 Time Sharing System Version 8.1 System Service Routines (File No. S360-31) Printed in U.S.A. GY28-2018-3

GY28-2018-3