

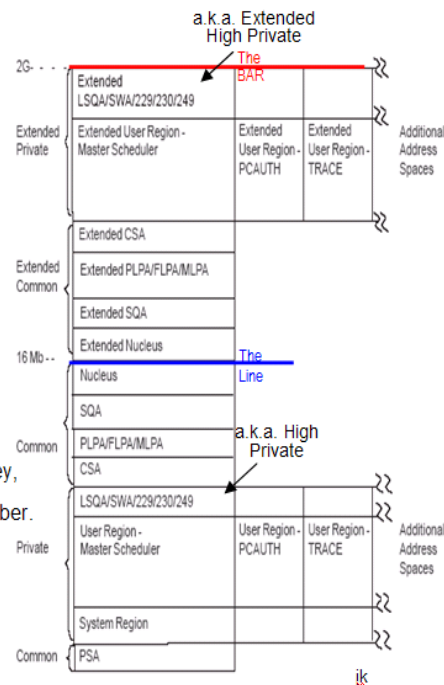


Why are Control Blocks Required?

Control blocks contain the management of the operating system and execution units (applications) that are executing. Each subsystem along with units of work (UOW) provides and use control blocks in order to interface and provide status information to other operating system components. They provide all debugging functionality in the event an operating system component or user application fails while the system is running. Control blocks serve as the anatomy of the operating system.

Control blocks provide isolation and integrity between executing components by where they are located in storage. The various virtual storage areas are displayed below and the location of a control block, whether it is in common or private storage, inherits its isolation characteristics from other control blocks.

Common storage is shared and used by all address spaces in the system. Because this area is used by any number of tasks in any number of address spaces concurrently, it is not normally used to make user modifications, but if they are they need to be serialized since this area provides universal information affecting the integrity and execution of many tasks including system tasks. There is one common area per LPAR or system image. Therefore, control blocks that are created and used from this storage offers information services for all tasks. The storage areas for common are Nucleus, System Queue Area (SQA), Link Pack Area (LPA) and Common Storage Area (CSA). Common is used below and above the sixteen megabyte line which is the addressing boundary between 24 and 31 bit addresses. Any control blocks that are created and used above the sixteen megabyte line are referred to as extended storage. Therefore, the extended area for SQA would be ESQA, LPA (ELPA). NOTE- The z/OS nucleus straddles the sixteen megabyte line. The **Private area** is formed by the following areas: Subpools 229, 230, and 249. This area allows private storage to be obtained in the requestor's storage protect key. The area is used for control blocks that can be obtained only by authorized programs (in z/OS) having appropriate storage protect keys. A **subpool** is a virtual storage area with the same properties regarding storage key, pageable or fixed, private or common, fetch protected or not, and so on. When a program *GETMAIN*s virtual storage addresses, it must indicate the subpool number. **Local System Queue Area (LSQA)**: This area contains tables and control blocks queues associated with the address space. LSQA is intermixed with SWA and subpools 229, 230, and 249 downward from the bottom of the CSA into the unallocated portion of the private area, as needed. ELSQA is also intermixed, But is allocated downward from 2G into the unallocated portion of the extended private area, as needed. LSQA does not take space below the top of the highest storage currently allocated to the user region.



Control blocks can be located in both real and virtual storage and contribute to the size of what is known as working-set for an address space and on a system wide level the Multi-Programming Level (MPL). Control blocks are pageable, although if they serve a universal level or are frequently accessed can be staged in real storage as demand dictates.

Private storage is acquired and used by processes or address spaces that provide a particular function or execution environment. The control blocks that are created and used within private storage offer unique information to processes that are running. As an example, all processes are associated to a Task Control Block (TCB), therefore TCB control blocks are created within the private storage area. An address space control block (ASCB) with its extension containing many TCBs is acquired from common storage. Since each address space has its own set of control blocks they can not access the private area of the other, hence providing storage integrity.

When are they created?

Control blocks are created from Initial Program Load (IPL) time (or Master Scheduler Initialization) through system image termination. They are created as a result of the Virtual and Real Storage Managers (VSM and RSM respectively). Control blocks are linked together in logical chains using pointers. A pointer is a field in

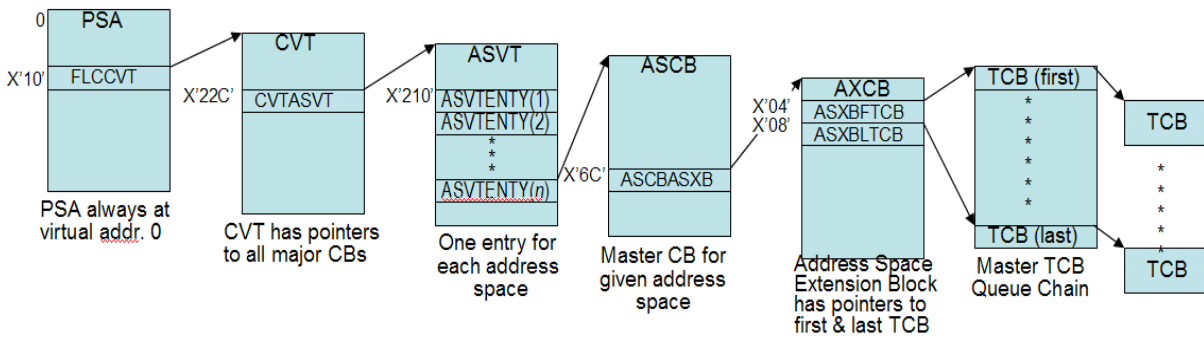


the control block which contains the address of the linked control block. Most z/OS control blocks can be located by following a series of pointers starting from the CVT (Communications Vector Table), which itself is always pointed to by the address at offset 16 (hex 10) within the Prefix Storage Area or PSA.

Exploring examples of control blocks

When a chain of control blocks needs to be constructed (i.e. to represent a queue of outstanding requests for a resource), each control block in the chain will usually contain a forward pointer to the next one and a backward pointer to the one before it.

Control blocks for application users in general are generated during address space creation (ASCB), during task management (TCB), or when requesting z/OS Services (Service Request Block) - as examples: an I/O, during an enqueue or lock management, interrupt handling, dispatching logic, program management, wait and post logic. Even during abend processing or system termination time. The user can also create control blocks, as an example during a GETMAIN Macro used for application programming requirements. In general, they are created during any event to manage and account for activity occurring within the operating runtime environment and are the tiepins that hold the operating system in unison.



Every component whether hardware or software plays a role in creating control blocks. They are not all created or destroyed at the same time, but are generated on demand depending on the function or event and terminated when the activity for that function ceases.

When an address space is created, major control blocks for an address space are constructed such as an ASCB (address space control block), ASXB (address space extension block) and an ASVT (address space vector table).

ASCB contains information and pointer addresses required for the management of this address space. Each address space is represented by an ASCB within the z/OS operating system and located in the System Queue Area (SQA). The reason why ASCBs are generated out of the SQA is due to interrupt processing. When an I/O interrupt happens, the CPU is switched away from the current address space to the interrupt handler. At that point, the interrupt routine gains hold in order to find the highest priority unit of work in an address space, the dispatcher has to scan through the ASCB chain and thus we see the need for ASCBs to be visible to all address spaces in the system. Some of the important and useful information in the ASCB:

- Identification Number called the "ASCBASID"
- Sequence numbering representing the ASCB's position on the dispatching queue "ASCBSEQN"
- Address of next/previous ASCB on ASCB Ready-Chain (ASCBFWD/ASCBBWD)
- Whether the address space is swapped in or out (ASCBFLG1)
- Dispatching priority of this address space (ASCBDP)
- Real storage frames allocated for this address space
- Pointer to JOBNAME Field for initiated programs (ASCBJBN)

Continued.....



Common Name: ADDRESS SPACE CONTROL BLOCK
Macro ID: IHAASCB
DSECT Name: ASCB
Owning Component: SUPERVISOR CONTROL (SC1C5)
Eye-Catcher ID: ASCB
Offset: 0
Length: 4
Storage Attributes: Subpool: 245
Key: 0
Residency: Below 16M
Size: 384 bytes
Created by: IEAMSWCB, IEAVEMRQ
Serialization: Serialization of the ASCB is dependent on the field being referenced. Some serialization techniques used here are local lock, compare and swap (CS), compare double and swap, and global intersect.
Function: Contain information and pointers needed for Address Space Control. The ASCB is non-swappable.

ASVT is used to keep track of all address spaces in the system image and there this control block does so, therefore it contains a list of all possible address space IDs, if assigned with associated ASCB. Remember an address space is created for each job, started task or mount request. The major control field here is the CVTASVT which this the location of the ASVT from the system's communication vector table. There is only one ASVT per system image (LPAR) located in the System Queue Area (SQA). Each address space has an entry in the ASVT except for the Master Scheduler's ASCB since it is hard coded in the firmware. The contents of this control block is made up of various slots.

- Number of free slots on the ASVT available (ASVTAAV)
- Address of first available slot on queue (ASVTRSHD)
- Maximum number of address spaces (ASVTMAXU)

Common Name: Address Space Vector Table
Macro ID: IHAASVT
DSECT Name: ASVT
Owning Component: Supervisor Control (SC1C5)
Eye-Catcher ID: ASVTASVT
Offset: 512
Length: 4
Storage Attributes: Subpool: 245
Key: 0
Residency: Below 16M
Size: Offset of ASVTEND minus offset of ASVTBEGN plus four times the value of ASVTMAXU.
Created by: IEAVNP09
Pointed to by: CVTASVT field of the CVT data area
Serialization: General CMS lock and dispatcher lock
Function: Mapping for the Address Space Vector Table

Continued...



ASXB control block also contains information about an address space but contains information of no particular interest to other address spaces or system components. This control block is located in the Local System Queue Area (LSQA) and a few important pointers are:

- Number of TCBs in this address space (ASXBTCBS)
- TCB dispatching queue for this address space (ASXBFTCB)
- IHSA (Interrupt Handler Save Area) pointer (ASXBIHSA)
- SRB dispatching queue for this address space (ASXBFSRB)

Common Name: Address Space Extension Block
Macro ID: IHAASXB
DSECT Name: ASXB
Owning Component: Supervisor Control (SC1C5)
Eye-Catcher ID: ASXB
Offset: 0
Length: 4
Storage Attributes: Subpool: 255
Key: 0
Residency: Below the 16M line
Size: Offset of ASXBEND minus the offset of ASXB
Created by: SYSGEN
IEAVEMIN
Pointed to by: ASCBASXB
Serialization: LOCAL lock
Function: Contains information and pointers needed for address space control. The ASXB is swappable, and the ASCB is not.

Dispatchable units of work[†] are represented by control blocks of two types - Task control blocks (TCBs) and Service request blocks (SRBs). TCBs represent tasks executing within an address space, such as user programs - but note that there are several TCBs associated with each address space, so more than one task could be running in any one address space at any one time (multi-tasking). SRBs represent "requests to execute a service routine" - they are usually initiated by system code executing from one address space to perform an action affecting another address space.

A task is a unit of work to the system. It competes for system resources and it is represented as a TCB. There is one TCB per task or process in the system for each unit of work. It is there for the operating system to monitor the activity. The TCBs are located in the LSQA. The TCB points to other control blocks that it requires for its existence.

The dispatching queues are chains of control blocks representing address spaces and units of work. The ASCB ready queue is a chain of the Address Space Control Blocks (ASCBs) of those address spaces which are swapped in and contain at least one TCB or SRB which is ready to execute (i.e. not awaiting the completion of any other event). The ASCBs are chained together in priority order - i.e. the ASCB with the highest dispatching priority is at the front of the chain. Each address space then has its own chain of ready SRBs and/or TCBs, pointed to from its ASCB. Whenever an event completes which changes the status of an address space, the relevant z/OS function updates the dispatching queues to reflect it.

[†] See #10 - zTidBits (Dispatchable UOW - Units of Work)

Continued...



TCB

Common Name: TASK CONTROL BLOCK
Macro ID: IKJTCB
DSECT Name: TCBFIX (DSECT card precedes prefix). The label, TCB, should be used in the USING statement for the TCB proper. TCBXTNT2 is the DSECT name for common extension.
Owning Component: Task Management (SC1CL)
Eye-Catcher ID: TCB
Offset: 256
Length: 4
Storage Attributes: Subpool: 253
Key: 0
Residency: Below 16 MB line
Size: 408 bytes
Created by: IEAMSWCB, ATTACH

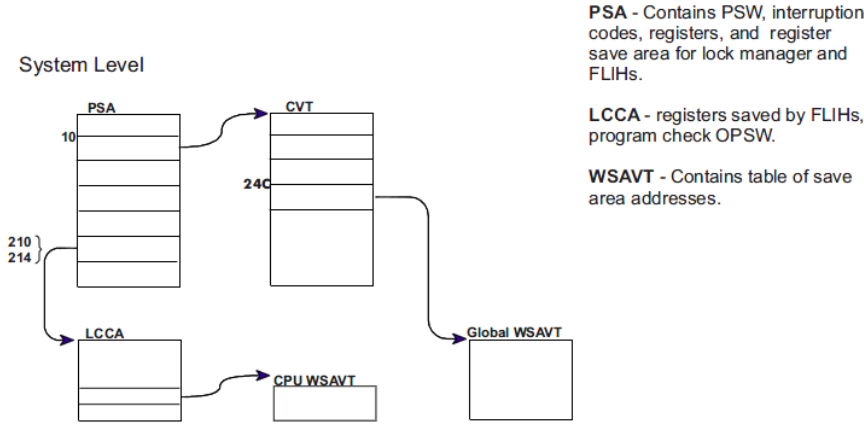
SRB

Common Name: Service Request Block
Macro ID: IHASRB
DSECT Name: SRBSECT
Owning Component: SUPERVISOR CONTROL (SC1C5)
Eye-Catcher ID: SRB
Offset: 0
Length: 4
Storage Attributes: Subpool: Common, Fixed Storage
Key: 0
Residency: ABOVE OR BELOW THE 16M LINE
Size: 44 BYTES
Created by: Control program routines
Pointed to by: Built and initialized in user-allocated storage and passed as a parameter to the SCHEDULE macro. Pointed to by register 0 on entry to the SRB routine whose address is in SRBEP.
ASCBXMPQ FIELD OF THE ASCB DATA AREA
ASXBFSRB FIELD OF THE ASXB DATA AREA
ASXBLSRB FIELD OF THE ASXB DATA AREA
IOSSRB FIELD OF THE IOSB DATA AREA
PCBSRB FIELD OF THE PCB DATA AREA
SRBFLNK FIELD OF THE SRB DATA AREA
SVTGSMQ FIELD OF THE SVT DATA AREA
SVTLSMQ FIELD OF THE SVT DATA AREA
SVTSRBA FIELD OF THE SVT DATA AREA
TQESRB FIELD OF THE TQE DATA AREA
TVCSSRBA FIELD OF THE TVCS DATA AREA
WEBUPTR field of the WEB data area
Serialization: Owner-serialized.
Function: Used as input to the SCHEDULE macro when scheduling a routine for asynchronous execution.

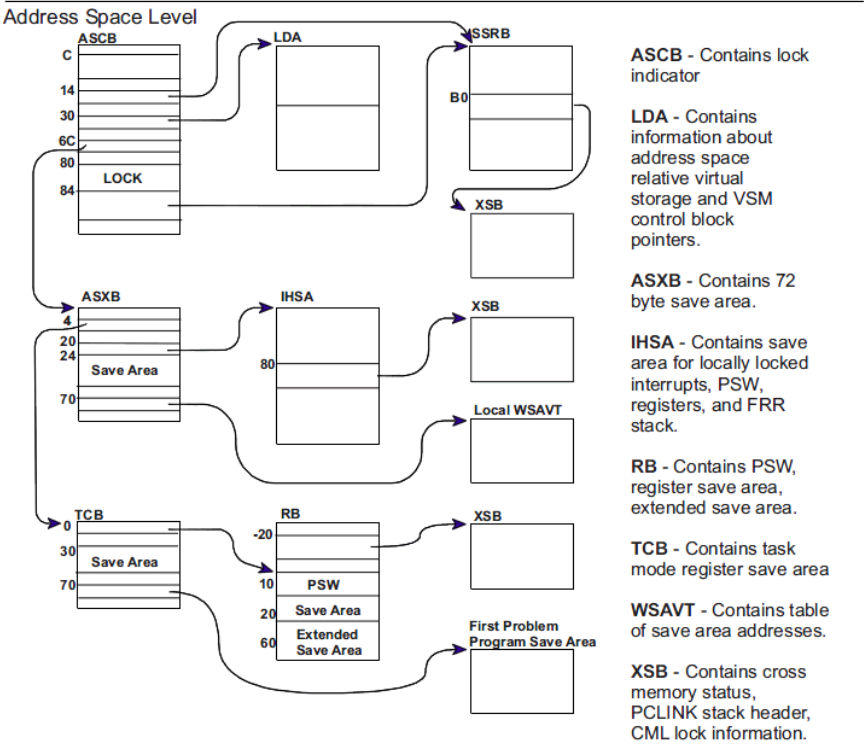
The next illustration incorporates additional control blocks, but are not described in detail.



- LCCA (Logical Configuration Communication Area)
- LDA (VSM Local Data Area)
- IHSA (Interrupt Handler Save Area)
- RB (Request Block)
- WSAVT (Work/Save Area Vector Tables)
- XSB (Extended Status Block)



- PSA** - Contains PSW, interruption codes, registers, and register save area for lock manager and FLIHs.
- LCCA** - registers saved by FLIHs, program check OPSW.
- WSAVT** - Contains table of save area addresses.



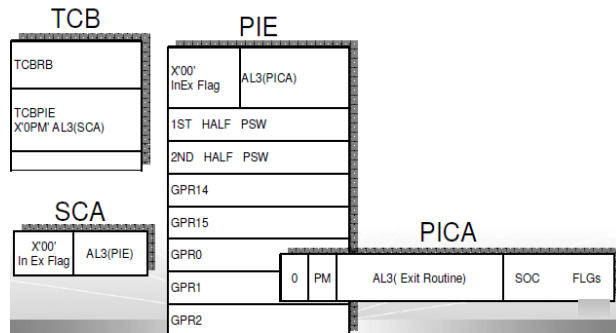
- ASCB** - Contains lock indicator
- LDA** - Contains information about address space relative virtual storage and VSM control block pointers.
- ASXB** - Contains 72 byte save area.
- IHSA** - Contains save area for locally locked interrupts, PSW, registers, and FRR stack.
- RB** - Contains PSW, register save area, extended save area.
- TCB** - Contains task mode register save area
- WSAVT** - Contains table of save area addresses.
- XSB** - Contains cross memory status, PCLINK stack header, CML lock information.

One of the important fields used by the TCB is the Specify Program Interrupt Element (SPIE). When a program check (abend) occurs control is given to the First Level Interrupt Handler (FLIH). FLIH will make a decision on who will get control such as either to the Control Program or a user-specified SPIE exit routine. The presence of a SPIE environment is reflected by a field in TCB (TCBPIE). This field is used for recovery termination, therefore if a SPIE environment exist, FLIH gives control to a SPIE EXIT. If no SPIE EXIT exist it is designated by all zeros in this field and a dump will be produced.



SPIE – Required Control Blocks

An example of when additional (dynamically created) control blocks are produced is when SPIE is invoked. The 3 major control blocks are the Program Interrupt Control Area (PICA), the Program Interrupt element (PIE) and the SPIE Control Area (SCA). These 3 control blocks establish the SPIE environment. A task is connected to this environment by a TCB pointer (TCBPIE). The TCBPIE field points to the SCA and the SCA points to the PIE. PIE in turns points to PICA.

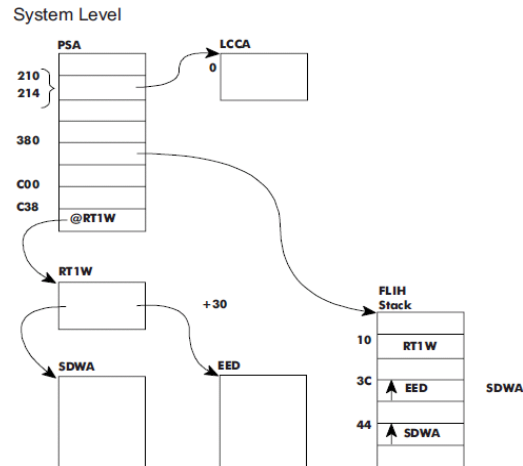


System diagnostic work area (SDWA) record When a software error occurs, the system gathers diagnostic information for the error and places it into a system diagnostic work area (SDWA) control block. A recovery routine can request that the system create a software-type record from the information in the SDWA and record it to the logrec data set.

SDWA information such as registers, PSW, locks held at the time of error, completion code, data describing reasons and conditions for entering the recovery exit routine, the CSECT in which the error occurred, module name, and FRR ID.

NOTE: The FRR (functional recovery routines) stacks are often useful for understanding the latest processes on the processors. They are mapped by the FRR control block and consist of a header and 16 20 byte FRR entries which are added and deleted dynamically as processing occurs. There is always one set of FRR stacks per processor. Look for the pointer to the current FRR stack at PSA +X'380' (PSACSTK). This will tell you where to find the FRR that was current at the time an error occurred.

FLIH: Interrupt handlers are divided into two parts: A First-Level Interrupt Handler (FLIH) and the Second-Level Interrupt Handlers (SLIH). FLIHs are also known as hard interrupt handlers or fast interrupt handlers, and implements at minimum platform-specific interrupt handling similarly to interrupt routines. In response to an interrupt, there is a context switch, and the code for the interrupt is loaded and executed. The job of a FLIH is to quickly service the interrupt, or to record platform-specific critical information which is only available at the time of the interrupt, and schedule the execution of a SLIH for further long-lived interrupt handling.



Input – Output Control Blocks

Before an I/O process can begin, a user program must OPEN the dataset to be accessed. The OPEN process builds the DCB (data control block), which contains various information about the dataset, including the address of the access method to be used to process it. It also builds a DEB (data extent block) containing further information about how the dataset is to be processed, including, for DASD datasets, the device address and the physical addresses of the extents of the dataset. These control blocks are used in the processing of subsequent I/O instructions for the dataset.

The access method builds more control blocks - the IOB (input output block) and the ECB (event control block). The IOB contains information required by the next stage of the operation (EXCP processing), and pointers to all the other control blocks associated with the I/O operation. Most importantly, it points to the channel program. This is also built by the access method, and consists of a series of channel command words (CCWs) which describe the I/O operations to be performed by the channel subsystem, including details such as the address of the area of storage into/from which data is to be transferred and the amount of data to be transferred. Finally, the access method issues the EXCP macro instruction, which invokes an SVC, causing an



interrupt, and the interrupt handler passes control to the EXCP processor. The next time the access method is dispatched it will issue a WAIT macro against the ECB representing this I/O request.

EXCP (Execute Channel Program) builds a control block called the I/O Supervisor Block (IOSB), fixes the page containing the storage to be read/written by the I/O operation so that it cannot be paged out before the operation is complete, and translates the addresses in the channel program from virtual to real addresses, as only real addresses are meaningful to the channel subsystem. It then invokes the STARTIO macro instruction to activate the I/O Supervisor (IOS) to process the I/O.

IOS builds an IOQ (Input/output queue) control block and ORB (Operation Request Block) describing the I/O for the channel subsystem, obtains control of the device (i.e. preventing any other concurrent I/O requests to it) by obtaining the relevant UCB lock (Unit Control Block), and then issues a SSCH (start sub channel) instruction to request the channel subsystem to perform the I/O operation.

The channel subsystem executes the channel program, which causes the requested I/O operation to occur, and it transfers the data being input (output) into (out of) the real storage locations referred to in the channel program. It updates control blocks including the IRB (Interrupt Response Block), which holds the return code from the I/O operation and can be interrogated to check whether completion was successful or not. It then posts an I/O interrupt to indicate the completion of the request.

The I/O interrupt handler schedules a SRB to run the "IOS post status" routines. These check the status of the completed I/O, invoke error recovery routines if required, and for successful I/Os, return control to EXCP. EXCP will POST the ECB which was created by the access method to indicate the completion of the I/O. The access method will therefore be made dispatchable, and when it regains control it will resume execution, returning control to the user program. The I/O operation is now complete and the user program can continue processing.

Control Block Life Cycle

The active life cycle of an address space will incur hundreds of more control blocks as it passes through various stages of aging including lack of activity. When this occurs paging will invoke buffering control blocks (page movement) to be generated in order to keep a working set trim in storage for new work to arrive, then be paged backs in for new real address assignments to process the newly arrived workload. Various control blocks are created by virtual and real storage management (VSM/RSM) as paging demands.

A few other examples when control blocks are created is during an LPAR activation, subsystem initialization, JES job submission/termination (or STARTED TASK -STC) , job (STC) execution, device allocation, file creation, any I/O such as sorting, statistics collection - SMF, abend / diagnosis processing , monitoring - RMF, recovery and communication management. As messages are written to the console or joblog control blocks are used to buffer the messages. The z/OS - MVS Data Areas Volumes provide in depth constructs and information on all the control blocks.

Mapping of Control Blocks to Virtual Storage Areas using Subpools

Control Blocks are constructs in storage areas within various subpools (0 - 255). The subpool provides the characteristics of the control block such as common or private storage, fixed or pageable, fetch protected or not fetch protected as well as protect keys and storage location. In a request for virtual storage, a subpool number indicates the type of storage that is requested. The subpool storage attribute also designates when the storage is freed, i.e. job-step task termination. *See each control block layout for its subpool storage attribute.*

Note: If no subpool number is designated, the virtual storage is allocated from subpool 0, which is created for the job step by the system when the job-step task is initiated.

Continued...



Subpool Decimal (Hex)	Location	Fetch Protection	Type	Storage Built Top-Down or Bottom-Up
0-127 (0-7F)	Private low	Yes	Pageable	B
129 (81)	Private low	Yes	Pageable	B

For purposes of control and virtual storage protection, the system considers all virtual storage within the region in terms of 4096-byte blocks. These blocks are assigned to a subpool, and space within the blocks is allocated to a task by the system when requests for virtual storage are made. When there is sufficient unallocated virtual storage within any block assigned to the designated subpool to fill a request, the virtual storage is allocated to the active task from that block. If there is insufficient unallocated virtual storage within any block assigned to the subpool, a new block (or blocks, depending on the size of the request) is assigned to the subpool, and the storage is allocated to the active task. The blocks assigned to a subpool are not necessarily contiguous unless they are assigned as a result of one request. Only blocks within the region reserved for the associated job step can be assigned to a subpool.

Subpool Decimal (Hex)	Location	Fetch Protection	Type	Storage Built Top-Down or Bottom-Up	Subpool Decimal (Hex)	Location	Fetch Protection	Type	Storage Built Top-Down or Bottom-Up	Subpool Decimal (Hex)	Location	Fetch Protection	Type	Storage Built Top-Down or Bottom-Up
130 (82)	Private low	No	Pageable	B	226 (E2)	Common SQA/ESQA	No	Fixed	T	244 (F4)	Private Low	No	Pageable	B
131 (83)	Private low	Yes	Pageable	B	227 (E3)	Common CSA/ECSA	Yes	Fixed	T	245 (F5)	Common SQA/ESQA	No	Fixed	T
132 (84)	Private low	No	Pageable	T	228 (E4)	Common CSA/ECSA	No	Fixed	T	247 (F7)	Common ESQA	Yes	DREF	T
203 (CB)	Private ELSQA	No	DREF	T	229 (E5)	Private high	Yes	Pageable	T	248 (F8)	Common ESQA	No	DREF	T
204 (CC)	Private ELSQA	No	DREF	T	230 (E6)	Private high	No	Pageable	T	249 (F9)	Private high	No	Pageable	T
205 (CD)	Private ELSQA	No	DREF	T	231 (E7)	Common CSA/ECSA	Yes	Pageable	T	250 (FA)	Private low	Yes	Pageable	T
213 (D5)	Private ELSQA	Yes	DREF	T	233 (E9)	Private LSQA/ ELSQA	No	Fixed	T	251 (FB)	Private low	Yes	Pageable	B
214 (D6)	Private ELSQA	Yes	DREF	T	234 (EA)	Private LSQA/ ELSQA	No	Fixed	T	252 (FC)	Private low	No	Pageable	B
215 (D7)	Private ELSQA	Yes	DREF	T	235 (EB)	Private LSQA/ ELSQA	No	Fixed	T	253 (FD)	Private LSQA/ ELSQA	No	Fixed	T
223 (DF)	Private ELSQA	Yes	Fixed	T	236 (EC)	Private high	No	Pageable	T	254 (FE)	Private LSQA/ ELSQA	No	Fixed	T
224 (E0)	Private ELSQA	Yes	Fixed	T	237 (ED)	Private high	No	Pageable	T	255 (FF)	Private LSQA/ ELSQA	No	Fixed	T
225 (E1)	Private ELSQA	Yes	Fixed	T	239 (EF)	Common SQA/ESQA	Yes	Fixed	T					
					240 (F0)	Private low	Yes	Pageable	B					
					241 (F1)	Common CSA/ECSA	No	Pageable	T					

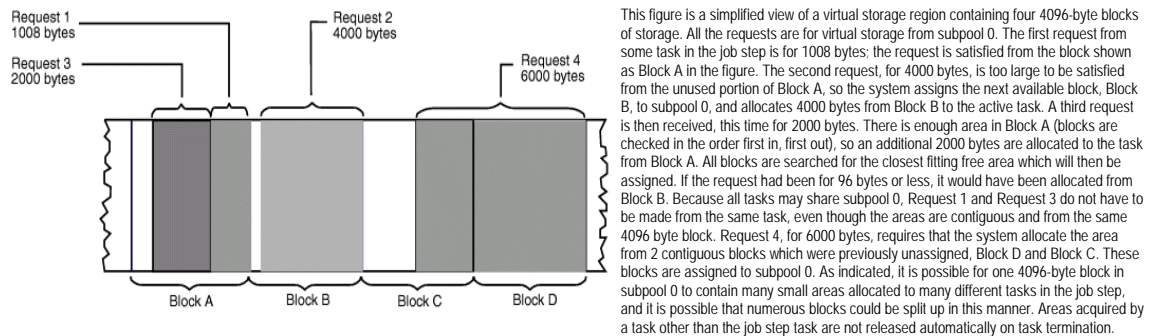
Continued....

Any subpool can be used exclusively by a single task or shared by several tasks. Each time that you create a task, you can specify which subpools are to be shared. Unlike other subpools, subpool 0 is shared by a task and its subtask, unless you specify otherwise. When subpool 0 is not shared, the system creates a new subpool 0 for use by the subtask. As a result, both the task and its subtask can request storage from subpool 0 but both will not receive storage from the same 4096-byte block. When the subtask terminates, its virtual storage areas in subpool 0 are released; since no other tasks share this subpool, complete 4096-byte blocks are made available for reallocation.



Note: If the storage is shared, it is not released until the owning task terminates.

When there is a need to share subpool 0, you can define other subpools for exclusive use by individual tasks. When you first request storage from a subpool (GETMAIN) other than subpool 0, the system assigns new 4096-byte blocks to that subpool, and allocates storage from that block. The task that is then active is assigned ownership of the subpool and, therefore, of the block. When additional requests are made by the same task for the same subpool, the requests are satisfied by allocating areas from that block and as many additional blocks as are required. If another task is active when a request is made with the same subpool number, the system assigns a new block to a new subpool, allocates storage from the new block, and assigns ownership of the new subpool to the second task.



You can customize your z/OS environment by using an assortment of exit routines that have access to various control blocks available in the environment from which it was called.

CAUTION: Because some exit routines run fully authorized, they are free to alter any field in any control block to which it has access. These exit routines can also generate their own control blocks by using a GENCB macro.

Control Block Access

You can find a control block if you know:

- a) The address of another control block containing a pointer to the control block you want.
- b) Where in the second control block that pointer is held.

More than likely, you will have to trace through several control blocks to find the block you want.

Once you decided which control block you need to access, you will need to work out the route to take to find it.

Clearly you must start from a known point and usually you will have two main choices for your starting point:

1. An address supplied to your program by its caller. It is common for z/OS exits in particular to be supplied with addresses of control blocks they may wish to access, and the documentation for the exit point may explain how to find other relevant control blocks if required.
2. The Communications Vector Table (CVT) is the starting point for most of z/OS' control block chains. Its address is always held at offset 16 (Hex 10) into the Prefix System Area (PSA), and since the PSA can always be found at virtual address zero, this provides a fixed point from which to start tracing a chain.

Example – Beginning with the CVT:

