

IBM z/VSE  
VSE Central Functions



# Supervisor Diagnosis Reference

*z/VSE 4.2*



IBM z/VSE  
VSE Central Functions



# Supervisor Diagnosis Reference

*z/VSE 4.2*

**Note!**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

**First Edition 07/11/07**

© Copyright International Business Machines Corporation 1985, 2008. All rights reserved.  
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Notices</b> . . . . .	vii
Programming Interface Information . . . . .	vii
Trademarks and Service Marks . . . . .	vii
<b>Preface</b> . . . . .	ix
Related Publications . . . . .	x

---

<b>Chapter 1: System Layout</b> . . . . .	1
z/Architecture Mode and 64-Bit Real Addressing . . . . .	3
Address Space Layout . . . . .	5
Storage Layout (Real and Virtual Address Spaces) . . . . .	7
Real Storage Layout . . . . .	9
Partition Layout . . . . .	11
SVA (Shared Virtual Area) Layout . . . . .	12
Minimum System GETVIS Requirements as Calculated by IPL . . . . .	14
System GETVIS Requirements when Vendor Exits are Enabled . . . . .	17

---

<b>Chapter 2: Design Information</b> . . . . .	19
<b>Interrupt Processors</b> . . . . .	21
I/O Interrupt . . . . .	21
Program Check Interrupt . . . . .	22
External Interrupt . . . . .	27
Machine Check Interrupt . . . . .	27
Supervisor Call Interrupt (SVC) . . . . .	27
<b>Dispatcher</b> . . . . .	29
z/VSE (Turbo) Dispatcher - Introduction . . . . .	29
Turbo Dispatcher . . . . .	37
Steps to Task Selection . . . . .	50
VSE/AF Dispatcher - Internal Gating Mechanism . . . . .	55
VSE/AF Dispatcher - Time Slicing (Partition Balancing) . . . . .	66
z/VSE Dispatcher - Task Termination . . . . .	73
z/VSE Dispatcher - System Dump Interfaces . . . . .	77
z/VSE Dispatcher - VSE/ICCF Support . . . . .	86
z/VSE Dispatcher - Partition Preparation and Cleanup . . . . .	88
VSE/AF Dispatcher - Identifiers and Limits . . . . .	93
<b>Physical Input/Output Control System (PIOCS)</b> . . . . .	97
I/O Request Enqueuer . . . . .	97
Scheduler . . . . .	100
I/O Interrupt Handler . . . . .	101
I/O Error Processing . . . . .	110
Disk Error Recovery . . . . .	113
Resident Tape Error Recovery . . . . .	114
ERP Message Writer . . . . .	115
Missing Interrupt Handler . . . . .	116
<b>Lock Management</b> . . . . .	117

LOCK and UNLOCK (SVC 110 - X'6E')	118
Lock Manager Internals	122
Deadlock Detection	125
DASD Sharing (Lock Manager)	129
Service and Debugging Information	134
<b>Channel Program Translation</b>	<b>161</b>
<b>Page Management</b>	<b>189</b>
General	189
Control Block Allocations	195
PMRAS space layout	199
Data Structures of Page Management	199
Page Faults and Page Frame Selection	215
Page Handling Routines	221
Load Leveling	250
<b>Storage Management</b>	<b>257</b>
General	257
Dynamic Storage Allocation	263
Address Space Layout and GETVIS Areas	263
GETVIS Processing	264
z/OS (OS/390) Storage Management Services	287
<b>z/Architecture Cross Memory Communication</b>	<b>297</b>
Description	297
The Cross Memory Environment	297
Cross Memory Services	299
Cross Memory Terminology	300
Termination Processing - Service User	301
Termination Processing - Service Provider	301
Control Register Save Area	301
Control Register Save Area Initialization	302
Task Interrupt Handling	302
<b>z/Architecture Subsystem Storage Protection</b>	<b>305</b>
Description	305
<b>z/Architecture Access Registers</b>	<b>307</b>
Introduction	307
Address Spaces	317
Data Spaces	318
<b>z/Architecture Linkage Stack</b>	<b>325</b>
Introduction	325
Linkage Stack - z/Architecture implementation	325
\$IJBLSTK - Create/Modify/Delete linkage stack	326
<b>Capacity Measurement Tool (CMT) in z/VSE 4.1</b>	<b>329</b>
Introduction	329
Characteristics of the CMT system task	329
System Resources	329
New Macro	329

<b>Program Retrieval</b> . . . . .	331
DASD Sharing Environment . . . . .	360
Program Retrieval - Tape Fetch . . . . .	363
OS/390 Program Retrieval Services . . . . .	364
<b>Machine Check, Channel Check and CRW Handling</b> . . . . .	367
<b>Job Accounting</b> . . . . .	375
<b>Software Re-IPL</b> . . . . .	379
<b>Console Support</b> . . . . .	383
<hr/>	
<b>Chapter 3: Diagnostic and Debugging Aids</b> . . . . .	397
<b>Diagnostic Aids</b> . . . . .	399
Fixed Storage Locations in Processor Storage (Low Core) . . . . .	400
Supervisor Patch Area . . . . .	401
Phase Load Trace Table . . . . .	402
Hard Wait Codes . . . . .	403
<b>Cancel Code to Message Code Cross-Reference</b> . . . . .	409
<b>Debugging Facilities</b> . . . . .	411
Features . . . . .	411
How to Find and Read the Debug Trace Area . . . . .	414
Format of the Debug Trace Entries . . . . .	416
DEBUG ON Command . . . . .	424
DEBUG TRACE Command . . . . .	425
DEBUG SELECT Command . . . . .	427
DEBUG SHOW Command . . . . .	428
DEBUG STOP Command . . . . .	429
LOCATE Command . . . . .	430
SHOW Command . . . . .	432
<hr/>	
<b>Appendices</b> . . . . .	433
<b>Appendix A. Supervisor Data Areas (without I/O)</b> . . . . .	435
Dynamic Class and System Limits Table (CLIMADR) . . . . .	439
Dynamic Class Table . . . . .	440
Space Control Block (SCB) . . . . .	442
Partition Control Blocks . . . . .	444
Task Control Blocks . . . . .	447
Save Areas . . . . .	449
Control Blocks related to Lock Management . . . . .	453
Event Control Block (ECB) . . . . .	454
Resource Control Block (RCB) . . . . .	455
VIO Control Blocks . . . . .	456
OS/390 Control Blocks . . . . .	461
<b>Appendix B. I/O Control Blocks</b> . . . . .	467
Basic Input/Output Control Words (z/Architecture) . . . . .	467

Input/Output Control Blocks and Areas . . . . .	469
Machine and Channel Check Control Blocks . . . . .	507
Track Hold Table (THTAB) . . . . .	512
<b>Appendix C. Samples . . . . .</b>	<b>513</b>
<b>Appendix D. XPCC/APPCVM Protocol . . . . .</b>	<b>519</b>
<b>Appendix E. Performance Monitoring Interface . . . . .</b>	<b>555</b>
Interactions . . . . .	555
<b>Index . . . . .</b>	<b>559</b>



---

## Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

---

## Programming Interface Information

This book documents information that is NOT intended to be used as a Programming Interface of z/VSE.

---

## Trademarks and Service Marks

The following terms, denoted by an asterisk (\*) in this publication, are trademarks of the IBM Corporation in certain countries:

ACF/VTAM	IBM
AFP	MVS/ESA
AT	NetView
CICS	PR/SM
CICS/VSE	SQL/DS
CUA	System/370
ECKD	System/390
Enterprise Systems Architecture/370	VM/ESA re-branded to z/VM
Enterprise Systems Architecture/390	VSE/ESA re-branded to z/VSE
ESA/370	VTAM
ESA/390	z/Architecture
ES/9000	OS/390 re-branded to z/OS
ESCON400	z/VSE
IBM System z	z/VM
z/OS	



---

## Preface

This manual is intended primarily for use by IBM personnel responsible for program service. It is one of three publications that describe the design and the internal control flow of the VSE/Advanced Functions Supervisor. The manual supplements the program listings by providing text and charts as follows:

- Chapter 1: System Layout

Provides general information about the VSE supervisor, its basic functions and modes, storage organization and storage allocations.

- Chapter 2: Design Information

Contains a detailed description of the various supervisor functions and components. These descriptions provide information necessary to become familiar with the internal logic of the supervisor.

- Chapter 3: Diagnostic and Debugging Aids

In this chapter information is provided which may be especially helpful in diagnosing program errors.

- Appendices:

A: Layout of commonly used supervisor data areas

B: Layout of commonly used I/O control blocks

C: Contains samples of track hold processing.

D: Contains the XPCC / APPCVM protocol.

E: Describes the interface for performance monitoring.

---

## Related Publications

The other two publications describing supervisor functions are:

- *z/VSE Supervisor Diagnosis Reference Error Recovery and Recording Transients*, SC33-6326
- *z/VSE Supervisor Diagnosis Reference Logical Transients and \$IJSxxx Phases*, SC33-6324.

For overall system logic, the following manuals are to be used in addition:

- *z/VSE Supervisor Diagnosis Reference Initial Program Load and Job Control*, SC33-6325
- *z/VSE Supervisor Diagnosis Reference Librarian*, SC33-6330
- *z/VSE Supervisor Diagnosis Reference Linkage Editor*, SC33-6328.

For efficient use of Diagnosis Reference publications, the reader should be familiar with the information contained in:

- *z/Architecture Principles of Operation*, SA22-7832
- *z/VSE Guide to System Functions*, SC33-8312
- *High Level Assembler Language Reference*, SC26-8265.

Procedures for isolating problems and analyzing storage dumps are contained in:

- *z/VSE Planning*, SC33-8301
- *z/VSE Diagnosis Tools*, SC33-8313
- *VSE/ESA Extended Addressability*, SC33-6724.

---

## Chapter 1: System Layout

The SUPERVISOR is that part of the VSE system which controls the execution of programs and which provides common services for them. The supervisor consists of:

- The **Supervisor nucleus** (fixed)
- The **Supervisor transients**
- Several **SVA resident phases**

The supervisor nucleus and the SVA resident phases are loaded at IPL time, whereas transients, unless they execute from the SVA, are loaded as needed from the sublibrary IJSYSRS.SYSLIB.

The following labels define the supervisor storage locations which are preserved for the different types of transient routines:

LTA	(Logical Transient Area) (\$\$B..... Phases) Eyecatcher: 'B-TRANSIENT AREA'
PTA	(Physical Transient Area) (\$\$A..... Phases) Eyecatcher: 'A-TRANSIENT AREA'
RTA	(Ras Transient Area) (\$\$R..... Phases)

The major functions performed by the supervisor are:

- Interrupt processing
- Task dispatching
- Physical input/output control (PIOCS)
- Channel program translation
- Page management
- Storage management
- Program Call support
- Subsystem Storage Protection support
- Resource management
- Job accounting
- Program retrieval (FETCH or LOAD)
- Error recovery and recording
- Operator communication
- Common Supervisor Services (SVCs)
- Access Register support
- Data Space support
- Linkage Stack support

**Notes:**

1. z/VSE V4 is delivered with one supervisor, \$\$\$SUPI, only.

- It executes in z/Architecture mode.
- It provides support for 1024 devices.

z/VSE V4 does no longer provide supervisor generation options. You can still generate a supervisor to get a listing of the \$\$\$SUPI supervisor.

---

## z/Architecture Mode and 64-Bit Real Addressing

z/VSE V4 supports more than 2GB processor storage (up to 8GB in z/VSE 4.1). The size of a virtual address/data space is still restricted to 2GB. The real storage beyond 2 GB is managed by the page manager. It will be used as page frames backing virtual pages. The page manager will execute in 64-bit addressing mode whenever page frames are addressed that may be allocated above 2GB.

The 64-bit addressing mode may also be required for those system program that deal with real addresses beyond 2 GB (e.g. the machine check handle and initialization routines). All other programs, system programs as well as vendor and application programs, will continue to execute in 31-bit or 24-bit addressing mode only.

z/Architecture mode is a prerequisite to execute in 64-bit addressing mode.

IPL starts its execution in ESA/390 architecture mode, since this is the mode initially set by the hardware. It switches to z/Architecture mode after the supervisor and dispatcher are loaded. Once the supervisor receives control of the system, it executes in z/Architecture mode only.

### What does z/Architecture mode mean?

In z/Architecture mode the hardware works with

- 16-bytes PSWs
- 8-bytes general purpose registers
- 8-bytes control registers
- 4-bytes access registers
- prefix area that comprises two pages (8K)
- the layout and contents of the first page of the prefix area has changed, e.g.
  - the location of new and old PSWs
  - the location of addresses like
    - translation-exception identification
    - failing storage address
  - the store status save areas are now in the second prefix page

### Translation Tables

In ESA/390 architecture mode the hardware uses segment tables and page tables for address translation. In z/Architecture mode, the hardware uses region tables, segment tables and page tables to translate 64-bit addresses. In z/VSE, virtual address and data spaces are restricted to 2GB. Therefore, a space can still be mapped by a segment and page tables. No region tables are used.

### Emulation of ESA/390 Interrupt Information

All interrupts handled by the supervisor occur in z/Architecture mode, since the architecture mode is permanently changed to z before the supervisor is invoked the very first time. When executing in z/Architecture mode, the hardware uses the z/Architecture new / old PSWs and interrupt locations to handle interrupts. These are

- External Interrupt
- I/O Interrupt
- SVC Interrupt
- Machine Check Interrupt

- Program Check Interrupt

When the first level interrupt handlers get control, the interrupt information is available at the z/Architecture locations. However, throughout the supervisor (and related system / vendor phases), ESA/390 locations, especially ESA/390 old PSWs, are referenced. To avoid, that all these programs have to be changed, the first level interrupt handlers, pointed to by the z/Architecture new PSWs, emulate the ESA/390 old PSWs locations. This is possible, since the ESA/390 old and new PSW locations are not used by the hardware. Additionally, the program check handler emulates the Translation-Exception Identification at location x'90'. Since a program check or machine check can occur in 64-bit addressing mode and / or with a 64-bit address the machine check and program check handlers have been adapted accordingly.

## General Purpose Registers - Usage and Save Areas

z/VSE continues to work with 4-byte registers and 4-byte addresses -apart from a few strictly internal exceptions. This is true for real and virtual addresses.

- Virtual storage never exceeds 2GB.
- Virtual storage is PFIxed / TFIxed below 2GB. That means, the real address of a PFIxed/TFIxed storage location will always be a 31-bit address.
- z/VSE does not allow 64-bit mode and/or the use of 8-bytes registers for user applications.

The ESA/390 PSWs are emulated, so the existing save areas in z/VSE (8-byte PSW, 4-byte registers) are sufficient and have not been extended.

Whenever the system uses 8-bytes registers it has to ensure that no interrupt occurs.

## Control Registers - Usage and Save Areas

z/VSE only uses 4-byte control registers. Since storage is PFIxed below 2GB, the high order 4-bytes of a control register containing a real address are always zero. So the existing 4-byte control register save areas are sufficient and have not been extended.

## LRA Consideration

LRA for a PFIxed/TFIxed page will always return a 31-bit address. If LRA in 24-bit or 31-bit addressing mode returns a 64-bit address a special-operation exception is recognized. This can happen for pages that are not fixed and that are backed by page frames above 2GB. This special-operation exception is intercepted by the program check handler and passed to the page manager. The page manager provides a page frame below 2GB and returns a 31-bit address, which is passed to the issuer of the LRA.



## Address Space Layout

The following figure shows the standard address space layout.

- the shared area below 16MB (starting at '0') is named 'shared area (24-bit)'.
- the shared area at the end of the address space (may start below or above 16MB) is named 'shared area (31-bit)'.
- the shared area (31-bit) does always exist. The start is determined by the size of the private area.

The ESA hardware supports a segment size of 1 Megabyte. Since the hardware requires the boundaries between shared and private areas to be on a segment boundary, alignment takes place for both the shared area (24-bit) and the shared area (31-bit). The size for the private area (SYS PASIZE) as well as the size for shared partition allocation (SYS SPSIZE) has to be specified during IPL. This keeps the boundaries between shared and private areas static after IPL. Additionally the size for real partition allocation (SYS RSIZE) has to be specified during IPL, too. Therefore allocation of private, shared and real partitions can only be done within the fixed boundaries.

$\min(2 \text{ GB}, \text{PASIZE} + \text{shared areas})$

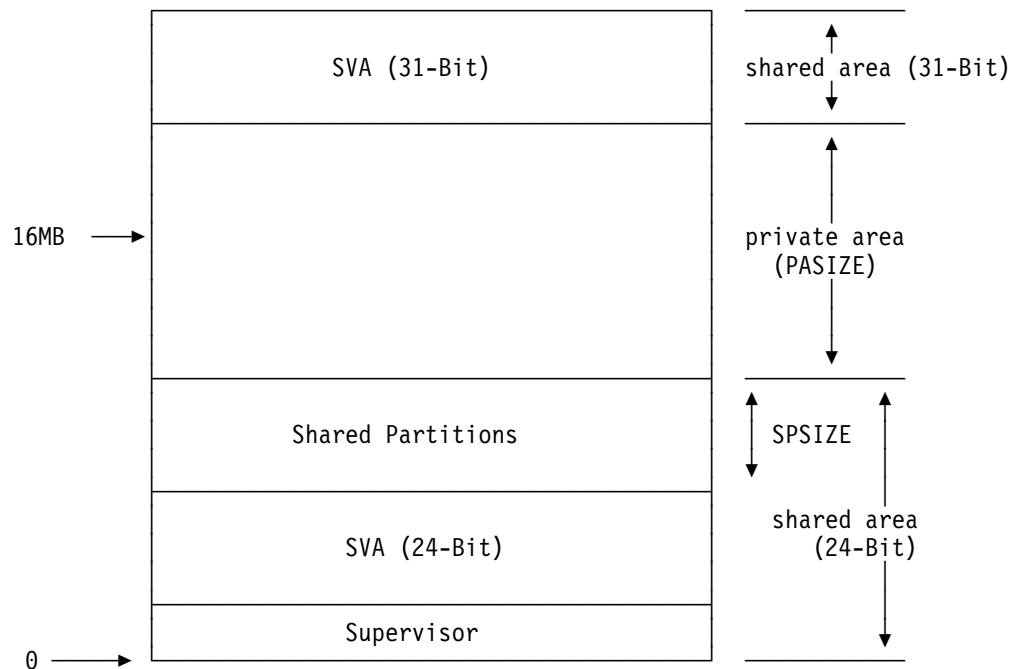


Figure 1. Address Space Layout(Address Space Size>16MB)

### Notes:

1. Even if there is no private area above 16MB (depending on PASIZE), the shared area (31-Bit) does exist.
2. The SMCOM control block and an EXTRACT service describe the storage layout.

**Shared Area (24-Bit)**

The shared area (24-Bit) comprises the following areas

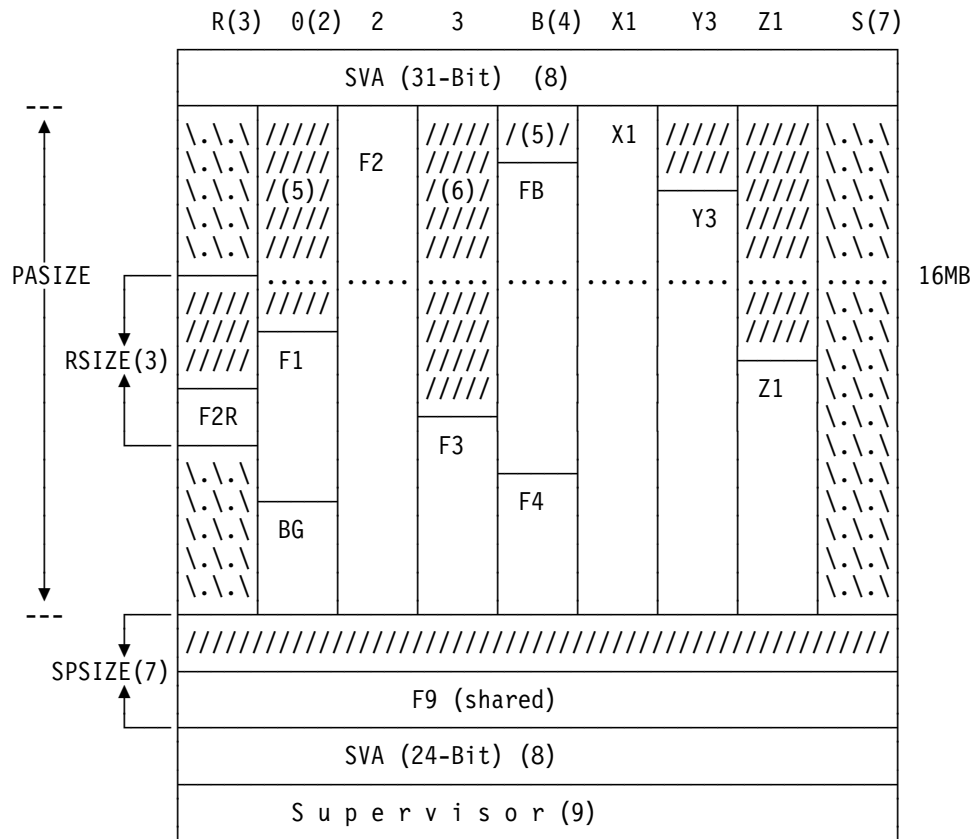
- Supervisor
  - including the SDAID trace area
- Shared virtual area (SVA) (24-Bit)
- Area for shared partition allocation

**Shared Area (31-Bit)**

The shared area (31-Bit) consists of the

- Shared virtual area (SVA) (31-Bit) only

## Storage Layout (Real and Virtual Address Spaces)



PASIZE = size of private area  
 RSIZE = size available for ALLOC R (may be zero)  
 SPSIZE = size available for ALLOC S (may be zero)  
 //// = invalid due to allocation  
 \.\.\ = always invalid

Figure 2. Storage Layout

1. The organization of a system with 6 static and 3 dynamic partitions is shown. The static partitions are allocated in the (static) virtual spaces 0, 2, 3 and B. There are 12 static spaces, 0,1,...,A and B where static partitions can be allocated.
 

Each static partition may or may not have a contiguous area of processor storage allocated for program execution in REAL mode.

The dynamic partitions are allocated in the dynamic spaces X1, Y3 and Z1.

An active virtual partition comprises at least 128KB in the virtual address area. The virtual partition size is always an integer multiple of

  - 64KB for static partitions
  - 1MB for dynamic partitions

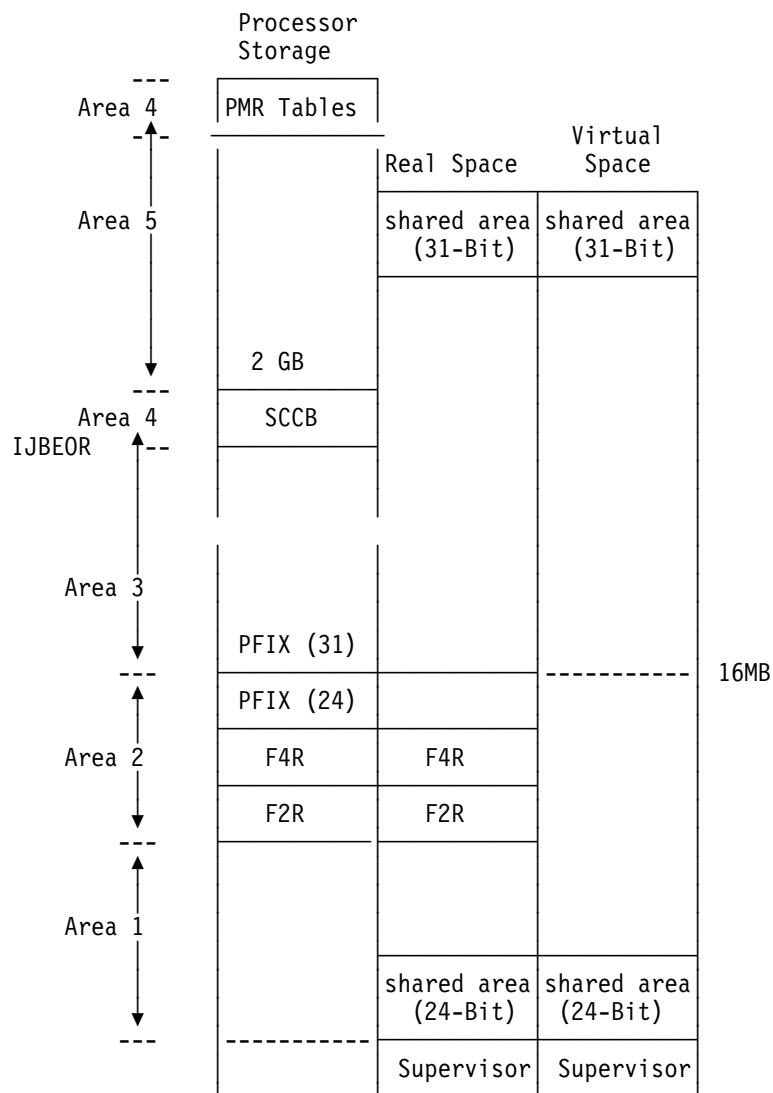
A partition may be allocated totally below 16MB or may cross the 16MB line. Note, that only one partition in an address space may cross the 16MB line. The virtual background partition (BG) is always active and is allocated in space 0.

The address area of an inactive virtual partition may be reduced to zero.
2. Space 0 does always exist. It is initialized by IPL with an initial BG size of 1MB, which can be changed explicitly later on by an ALLOC command.
3. Space R is used to allocate real partitions for real execution. Real allocation is restricted to an area below 16MB and the size for it must be reserved during IPL (SYS RSIZE). No addressability exists between real and private virtual partitions.
 

Allocations for real partitions must be an integer multiple of 4KB. Note, that real partitions are not required for PFIX processing. Instead the SETPFIX command can be used to reserve frames for PFIX processing.
4. The space id must be specified explicitly in an ALLOC request if more than one partition shall be allocated in one address space.
5. The area marked as 'invalid due to allocation' can be used to increase the existing partitions provided that the VSIZE is not exceeded. This is possible because the space has been created by specifying the space id explicitly and therefore page tables exist for PASIZE. Since the last partition (FB) crosses the 16MB line, no further partition can be allocated in the space.
6. If the space id was not specified, the area marked as 'invalid due to allocation' cannot be used, because page tables are allocated only for the initial allocation value. Otherwise, the area can be used to increase the existing partition or to allocate new ones, provided that VSIZE is not exceeded.
7. Shared partitions are allocated in a contiguous area following the SVA (24-Bit).The area available for shared partition allocation must be determined by SPSIZE (may be zero) and is restricted by VSIZE, too.
 

Tasks in shared partitions are dispatched in the shared space S.
8. For the layout of the Shared Virtual Area (SVA) see Figure 5 on page 12.
9. The supervisor area consists of the supervisor phase and areas dynamically allocated during IPL.
10. For the layout of a virtual and real partition see Figure 4 on page 11.

# Real Storage Layout



- EOR - Min(2GB, end of processor storage)
- Area 5 - Used for paging
- Area 4 - reserved for system use
- Area 3 - available for PFIX (31-Bit) (both user and system requests)
- Area 2 - defined by RSIZE (available for ALLOC R and PFIX (24-Bit))
- Area 1 - available for PFIX (24-Bit) (both user and system requests)

Figure 3. Real Storage Layout

**Note:** Frames in area 4 are never assigned to virtual addresses, whereas the real storage from address 0 up to IJBEOR is used for paging.

## Setting of Boundaries

The boundaries of area x, x=1,...,3, are described by the labels SMCOM.SMCRBGx and SMCOM.SMCRNDx. Area 2 does not exist if RSIZE=0 has been specified. Area 3 does always exist, however it may be used totally by the system, e.g. if it starts below 16MB. The boundaries are set as follows:

### Area 1

- $SMCRBG1 = SMCSSEND+1$
- $SMCRND1 = \text{MAX}\{\text{SMCSPEND}, \text{SMCRBG1}+\text{SMINSVPX}(\text{in bytes})-1, \text{SMCRBG3}-\text{RSIZE}(\text{in bytes})\}$ 
  - SMINSVPX - Minimum requirement for system PFIx in area 1

### Area 2

- $SMCRBG2 = SMCRND1+1$
- $SMCRND2 = SMCRBG3-1$
- Final RSIZE:  $\text{SIZE}(\text{SMCRBG2}, \text{SMCRND2})$

### Area 3

- $SMCRBG3 = \text{MIN}\{16\text{MB}, \text{IJBEOR}+1, \text{MAX}\{\text{area PFIxed by \$INTVIRT}, \text{SMINSPX3}(\text{in bytes})\}, \text{SMCSVA31}\}$
- $SMCRND3 = \text{IJBEOR} - \text{SIZE}(\text{minimum page pool})$ 
  - SMINSPX3 - Minimum requirement for system PFIx in area 3

Area 3 is always available and contains at least data PFIxed by \$INTVIRT (PMR tables for the shared area and for the first PMR address space).

### Area 5

Area 5 is only present with more than 2GB of processor storage.

- $SMCRBG64 = 2\text{GB}$
- $SMCRND64 = \text{SMCPEOR}$  (end of processor storage)

## Partition Layout

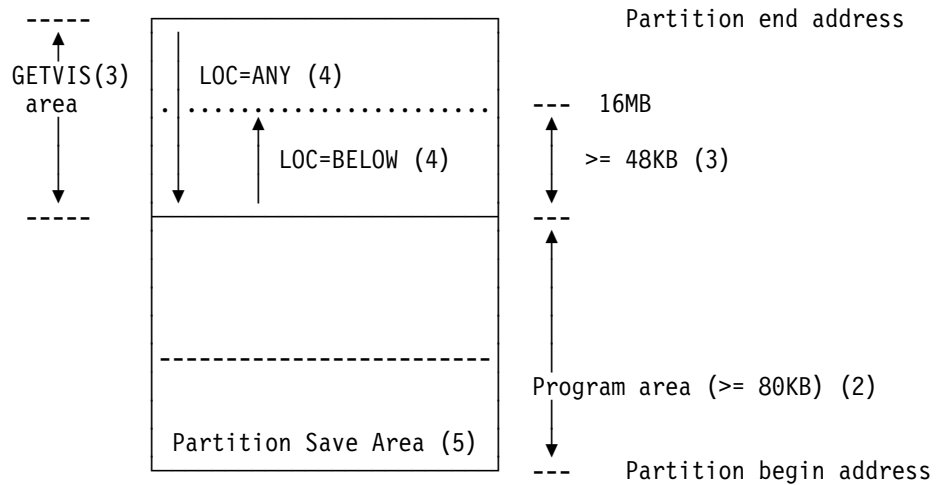


Figure 4. Partition Layout

1. Job start time, for the time stamp, is stored in the last 6 bytes of this area (bytes 82-87) when specified.
2. A partition always starts below the 16MB line with the program area, the size of which can be changed by the SIZE command or the EXEC ..,SIZE parameter. The program area is located totally below the 16MB line. If a partition crosses the 16MB line, the 31-bit area of the partition belongs to the partition GETVIS area.
3. A virtual partition always has a GETVIS area (minimum and default is 48KB). The minimum GETVIS area of 48KB is required below the 16MB line. In real mode the minimum/default value is 0KB; if the user wants to have a GETVIS area he must specify it implicitly by using the SIZE parameter in the EXEC statement.
4. GETVIS below the 16MB line can be requested with the GETVIS LOC=BELOW parameter. It is allocated bottom-up up to the 16MB line. GETVIS, that can be located anywhere in storage, can be requested with the LOC=ANY parameter. Storage is then allocated top-down. For partitions, that are located totally below the 16MB line, LOC=ANY is treated as LOC=BELOW.
5. In the partition save area the following information is stored:
  - Program Name (8 bytes)
  - Program Status Word (8 bytes)
  - General Registers 9-8 (64 bytes)
  - Job Start Time (1) (8 bytes)
  - Floating-point Registers (32 bytes)

## SVA (Shared Virtual Area) Layout

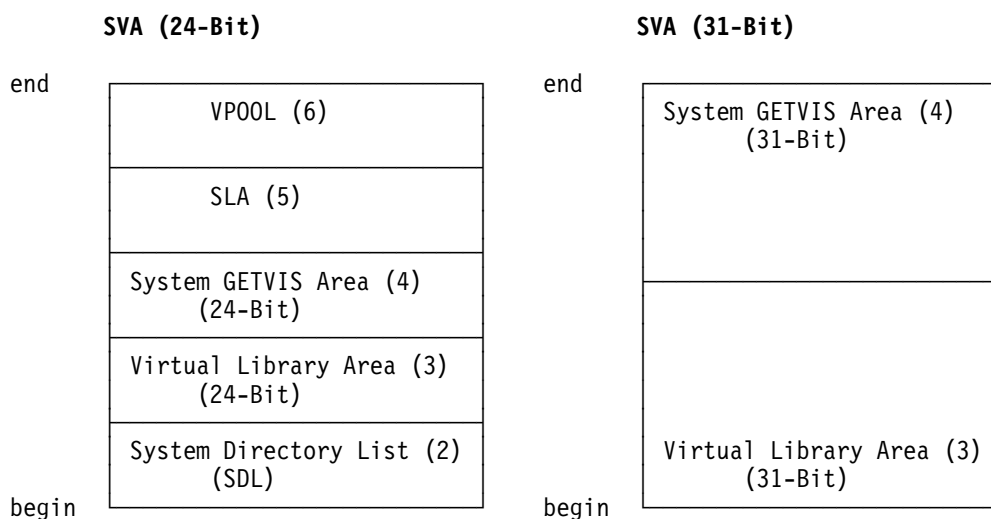


Figure 5. SVA Layout

1. Shared Virtual Area (SVA): An area where heavily used reentrant programs are loaded; they can be shared between partitions, and also parts of the system (e.g. End-of-Job processing routines). The SVA (31-Bit) consists of the VLA (31-Bit) and the system GETVIS area (31-Bit).
2. System Directory List (SDL): In-core directory of highly used programs (phases). For further information refer to "Shared Virtual Area (SVA)" on page 342. It is located in the SVA (24-Bit) and contains the entries for both the (24-Bit) and the (31-Bit) VLA.
3. Virtual Library (Phase Area): Contains highly used programs (phases) which can be shared between partitions and the system. For further information refer to "Shared Virtual Area (SVA)" on page 342. Depending on the RMODE of a phase, a phase is loaded in the VLA (31-Bit) (RMODE=ANY) or in the VLA (24-Bit) (RMODE=24).
4. The GETVIS area for the system can only be used by requestors with a storage protection key of zero. LOC=ANY users are served preferred from the system GETVIS area (31-Bit), whereas LOC=BELOW users are served from the system GETVIS area (24-Bit).
5. This area is allocated during IPL and used by Label Processing (SLA).
6. This area is allocated during IPL depending on the VPOOL parameter and is used as a buffer pool for VIO.



The following table shows the supervisor tables and buffers which are allocated at IPL time.

Copy buffers
Channel queue
CCW chains for DASD file protection
CCW chains for TAPE set mode
PUB2 areas
PUBX area
LUB extension
Pubscan tables
AVR table
Reentry rate table
Page frame table
Page to Disk Assignment String (PDAS)
Page table (1)
Segment table (1)
Page Table Assignment String (PTAS) (1)
Page-Out State List (POSL) (2)
Save areas for Access Registers
Console buffer
Hardcopy buffer
SYSREC buffer
PAGEIN table
Extended logout areas: IOEL, MCEL
Phase load lists for static partitions
VIO/VPPOOL area
Device control blocks
External interrupt buffer for IUCV/APPC/VM
Path ID table for IUCV/APPC/VM
Job accounting tables for static partitions
SAACOMM (supervisor/system dump communication area) (3)
Tasks data space control block (3)
Default job accounting table used during dyn. partition allocation
PCBATAB, PCEATAB, SMCBTAB, SCBATAB, PUBOWNER extension
SCBATAB, CPCBATAB
Linkage stack for static partitions except BG
PCB extension for static partitions except BG
Cross memory resource tables

Figure 6. Supervisor Areas, Allocated at IPL Time

1. Allocated for space 0, the shared space and the first page manager space.
2. Allocated for space 0 and the shared space.
3. Allocated for static partitions and AR.

## Minimum System GETVIS Requirements as Calculated by IPL

The size of the System GETVIS areas is the sum of three parts:

1. AF space requirements determined from supervisor generation parameters.
2. AF space requirements determined from IPL options, including Librarian requirements from \$INITCON.
3. Size specification from the GETVIS parameter of the SVA command. This is a user specification.

These sizes are used to calculate the total space allocated to the System GETVIS area. There is no space reservation before explicitly requested.

The system assumptions 1. and 2. are listed in Figure 7 to Figure 11 on page 16.

Acronyms:

```

dev  -- I/O devices
part -- partitions
sdev -- shared I/O devices
vdisk -- virtual disk(s)
/    -- per

```

<i>Figure 7. 24-Bit System GETVIS - Systemwide</i>		
<b>Item Algorithm</b>	<b>PFIX</b>	<b>Size</b>
SDAID buffers (SYS SDSIZE>0) (SYS SDSIZE=0)	no	90K 0K
Reentrant dump work area	no	72K
SLA work area <sup>1</sup>	no	12K
Hard copy support	no	5K
User save area (SYS JA=YES) (SYS JA=NO)	no	1K 0K
Librarian control blocks <sup>2</sup>	no	ca. 60K +7K/12 part

*Figure 8. 24-Bit System GETVIS - Static Partition Subpools*

<b>Item Algorithm</b>	<b>Subpool</b>	<b>PFIX</b>	<b>Size</b>
System control block address tables (PCBATAB, SMCBTAB, ...)	IINIT	24-bit	8K
Class table	IJBSSP	24-bit	4K
Subtask control blocks 6 subtasks per partition, 5 control blocks per page)	IPTIB	24-bit	60K
Subtask save areas (save area length 120 bytes, 6 subtasks per partition, 34 save areas per page)	IPSAV	no	12K
XECB subpool	ISXECB	no	4K
LOCK subpool (10 resources per partition, 3 owner elements per resource)	ILCKSP	no	12K
XTENT subpool	SPXTNT	no	4K
MSAT subpool	SPMSAT	no	4K
VIO subpool	ISPVIO	no	4K
Partition subpool (1 page per partition)	IJBBG, IJBFn	no	48K
Guaranteed DUMP subpool	IJBDMP	no	20K
Parameterized procedures (2 pages per partition)	IJBPRC	no	96K
SIO counter string (SYS JA=YES: IJBNSDEV*8*12) (SYS JA=NO: IJBNSDEV*4*12)	IJBFCB	24-bit	4K /42 sdev /85 sdev

*Figure 9. 24-Bit System GETVIS - Static Partitions (12)*

<b>Item Algorithm</b>	<b>PFIX</b>	<b>Size</b>
Security/Logging	no	4K
SYSFIL on FBA	no	4K
FETCH/LOAD trace tables (&AGPHLSL*12)	no	4K
LUBX for MSAT (4*(pgr_LUBs+system_LUBs*12))	no	16K
Job Control work areas <sup>1</sup> (2 pages per partition)	no	96K
Job Accounting tables (SYS JA=YES: (IJBNDDEV*6+ACCTALEN)*12) (SYS JA=NO)	no	19K /254 dev 0K

<i>Figure 10. 24-Bit System GETVIS - Dynamic Partitions</i>			
<b>Item Algorithm</b>	<b>Subpool</b>	<b>PFIX</b>	<b>Size</b>
Dynamic partition control blocks (4K per dynamic partition) (xy = LOGID)	IJBPxy	31-bit	(NPARTS-12)*4K
PUB ownership table (1 bit per device and partition)		no	1K /32 part, 254 dev

<i>Figure 11. 31-Bit System GETVIS</i>			
<b>Item Algorithm</b>	<b>Subpool</b>	<b>PFIX</b>	<b>Size</b>
GETVIS control information (minimum)	no	no	4K
XPCC buffers	no	31-bit	20K
Access lists (DUAL, PASENAL, 0,5K per partition, 8 lists per page)	IJBALE	31-bit	8K
Access module save areas (1K per partition)	no	no	12K
Data space control blocks (SCB, DSCB, ASTE, 256 bytes per vdisk, 16 blocks per page)	IJBDSP	31-bit	4K /16 vdisk
Virtual disk control blocks (128 + (352 bytes per vdisk), 11 blocks per page)	IJBVDI	31-bit	4K /11 vdisk
Attention Routine buffers	ARCHSMON ARCHPID	31-bit	32K
Attention Routine DEBUG buffers	SPDEBUG	31-bit	64K
Console router fixed queue space	IJBSCM IJBSCC	31-bit	48K
Console router pageable queue space	IJBSCM IJBSCC	no	124K
System control blocks	IINIT	31-BIT	32K
Dynamic partition control blocks (4K per dynamic partition) (xy=LOGID)	IJBPxy	31-bit	(NPARTS-12)*4K

<sup>1</sup> These areas are part of the 24-bit System GETVIS area, but their size is not reflected in the displayed values of the GETVIS command, because they are reserved from IPL to shut-down.

<sup>2</sup> The size of this area is calculated by the librarian phase \$INITCON.

---

## System GETVIS Requirements when Vendor Exits are Enabled

Since these system Getvis requirements cannot be considered by IPL, they have to be specified in the SVA command by the user.

<i>Figure 12. 31-Bit Pfixed System GETVIS</i>		
<b>Item</b>	<b>Subpool</b>	<b>Size</b>
TASKPROD/per task	PEXITM (maintask) PEXITS (subtask)	x'3A8'
IJBVEND control blocks	PEXITP	x'1A08'
PROEXCB/1 per exit	PEXITP	x'98'
vendor information (fixed) plus per vendor product	IPVEN	x'800' x'136'



---

## Chapter 2: Design Information

This chapter presents the design information by functions.

- Interrupt Processors  
The different interrupt types and supervisor routines to handle these interrupts.
- Dispatcher, Task Selection  
A description of the dispatching of system and user tasks.
- Physical Input/Output Control System (PIOCS)  
A description of device scheduling and I/O interrupt processing.
- Lock Management  
A description of the lock/unlock mechanism.
- CCW Translation and Retranslation  
A description of CCW-translation, retranslation and CCW fixing.
- Page Management  
Page fault handling, page manager services.
- Storage Management  
Short description of storage management routine.
- z/Architecture PC-ss support
- z/Architecture Subsystem Storage Protection Facility
- z/Architecture Access Register (translation and use)  
including
  - Data Space Support
- z/Architecture Linkage Stack
- Program Retrieval  
FETCH/LOAD operations including SVA usage.
- Machine- and Channel Check Recovery and Recording  
Types of machine checks, channel checks and resulting actions.
- Job Accounting  
Short description of job accounting routines.
- Software Re-IPL  
Short description of the Software Re-IPL routine.
- Console Support





---

## Interrupt Processors

The supervisor is designed to operate in z/Architecture mode on an IBM System z processor.

In the z/Architecture Program Status Word (PSW) bit 12 must be OFF.

Bit 12 of the PSW is the former Extended Control (EC) mode bit, which must always be ON in an ESA/390 PSW.

When the ESA/390 emulation code prepares the ESA/390 old PSW, bit 12 must be set ON, since the interrupted task is dispatched with an LPSW from the ESA/390 old PSW.

Processing may be interrupted by any of the following conditions:

- Input/Output Interruption
- Program Interruption
- Machine-Check Interruption
- Supervisor-Call Interruption
- External Interruption

An interruption condition consists in storing the current PSW as an old PSW, storing further detail information identifying the cause of the interruption, and fetching a new PSW (Refer to IBM z/Architecture Principles of Operation). Processing resumes with the ESA/390 emulation as specified by the new PSW prior to invoking the first level interrupt handler as specified in the former ESA/390 new PSW.

The first level interrupt handler saves all the information which is necessary to resume the interrupted processing at a later point in time. After initialization control is passed to the second level interrupt handler.

The second level interrupt handler which will be described in detail below performs the actual interrupt processing and after completion returns to the task selection routine.

---

## I/O Interrupt

Refer to Physical Input/Output Control System (PIOCS) later in this chapter.

---

## Program Check Interrupt

Like the other interrupt handlers on VSE, the program check handler gets control with a z/Architecture new PSW. As its first action, it emulates the ESA/390 PC old PSW at the ESA/390 PC old PSW location (storage location x'28'), and (if appropriate) the ESA/390 PER address at location x'98', the ESA/390 Translation Exception address at location x'90', and the Monitor Code at location x'9C'.

SDAID is called right after this prolog. SDAID will display only the rightmost 4 bytes of a register.

The program check handler inspects the program interruption code and passes control to the appropriate processing routine. It mainly differentiates between *programming* exceptions and *translation* exceptions. The program check handler is entered in real mode which means that the DAT-bit in the PSW is off.

## Handling of a Normal Program Check

If a normal program check is to be handled, the DAT bit in the current PSW is first turned on.

If the program check occurs in the supervisor code the system enters a hard wait, unless one of the following conditions is fulfilled:

- The supervisor failed due to incorrect input parameters passed by the user program. A list of addresses in the Supervisor is scanned to see if the program check occurred at any of these addresses. If so, the user program is canceled.
- A check is made to see if ACF/VTAM is active and executing an SVC 49 (X'31') or 53 (X'35') or one of its appendage routines. If so, that partition is canceled.
- A check is made to see if ICCF (SVC 82 - X'52') or an ICCF intercept routine is active. If so the ICCF partition is canceled.

If the system goes into a Hard Wait, SYSCOM bytes 4 through 7 and low-storage bytes 0 through 3 contain the appropriate hard wait code. The hardwait state is entered by means of the hardwait routine (label SYSERRORR). (see Chapter 5, Figure 169 on page 403).

If the program check occurred while the program check handler was active, the program check handler loads a hardwait PSW (X'000A0000 00001122') without modifying SYSCOM and low-storage 0-3. Furthermore an indicator is set to force other CPUs to enter hardwait whenever the program check handler is entered in a MP environment.

If the program check handler is entered due to a 'stack-full' exception, phase IJBLSTK is called to extend the linkage stack of the current task.

If the program check handler is entered due to a 'ALEN-translation-exception' because of an ALET 2, phase IJBLSTK is called to create a DUAL (if not available) and to initialize the third entry of the DUAL to address the home space of the currently active task.

If the program check occurs in a page handling overlap (PHO) appendage routine or in an I/O appendage routine, the interrupt status and general registers are saved in a separate save area (label SVPCSAVE) and the users program is canceled.

If the program check occurs in the problem Program, it will be canceled, unless a program check exit routine was specified. In this case the Program Check Handler passes control to a special routine at label PCROUT which saves the interrupt status information and general registers in the save area specified by the user's STXIT (PC) macro for the following purposes:

- To restore for continuation.
- To enable the user's PC routine to analyze the status.
- To facilitate analysis of a dump, should a dump be requested (the dump then contains all interrupt information).

To enter the user's PC exit routine, the PSW saved in SVEPSW is modified to point to the users PC exit routine and a special bit in the TCB is turned on, indicating that the PC exit routine is active. A program check encountered at a time while this bit is still on, causes the task to be canceled.

The user's PC exit routine must end with an SVC 17 (X'11' - EXIT PC) to resume processing at the point where it was disrupted. In this case the interrupt status information and general registers are restored to the program save area and the PC routine active bit in the TCB is reset.

Address-space-control element (ASCE) type exceptions as well as Region-First-Translation, Region-Second-Translation and Region-Third-Translation exceptions are handled like Segment-Translation exceptions: Depending on the RID either the user is cancelled or the system enters a hardwait with hard wait code x'FFA'.

## Handling of Special-Operation Exceptions from an LRA

A Special-Operation-Exception is raised when a LOAD REAL ADDRESS (LRA) instruction is executed in 24-bit or 31-bit addressing mode, and when bits 0-32 of the resulting real address are not all zeroes. This may happen when a virtual address of a page that is not PFIXED or TFIXED, but assigned to a page frame, has a corresponding 64-bit real address.

When this happens the program check handler calls the page manager to assign a page frame below 2 GB thus providing a real address below 2 GB. The program check handler then returns to the interrupted program and passes a real address which is below 2 GB.

For details on page manager processing please refer to description of the LRA Exception Appendage in chapter -- Heading 'LRAAPP' unknown --.

## Handling of Page Fault Interrupts

Page faults are a special type of program checks and are handled by an extension to the program check handler, the page fault first level interrupt handler (PFFLIH). By means of the RID (Routine identifier, label RID in the supervisor) it is determined what action is to be taken. Figure 13 on page 25 shows the various RIDs along with the actions taken if one of the appropriate routines causes a page fault.

The page fault handler also sets the TIBFLAG. This flag tells the dispatcher how to dispatch the task after the page fault has been handled. The TIBFLAG indicates that control is to be passed to SVRETURN if a supervisor service is to be reactivated.

If no page-fault appendage is provided for the interrupted task, a page fault request is queued for handling by the page management routines and the interrupted task is set not dispatchable (PMRBND).

If an appendage is present for the task, control is passed to the appendage, and the task causing a page fault remains dispatchable unless the page fault occurred during a supervisor service for the task. (Refer to *z/VSE System Macros User's Guide*, SC33-8236 for a more detailed explanation of Page Fault appendages.)

If, for a task owning an appendage, a page-fault-handling request has been queued previously, the pending request is not queued.

## Handling of Pseudo Page Faults

Pseudo-page-faults are a special type of program checks, that can occur in systems running under VM.

Pseudo-page-faults are page faults detected while processing on a virtual machine with I/O interrupts enabled and for which the SET PAGEX ON command has been issued. When these conditions are satisfied, VM causes a pseudo-page-fault exception by storing the virtual machine address that caused the page fault, reflecting a program interrupt to the virtual machine, and removing the virtual machine from page and execution wait. When VM has satisfied the page request for the virtual machine, it reflects a pseudo-page completion.

For both pseudo-page-fault exception and pseudo-page-fault completion, the VSE virtual machine is removed from the wait state by VM and given control by a program check interrupt.

A pseudo-page-fault is first tested to see if it is a completion. If this is the case and the page brought in is the same as the previous completion with no faults in between, control is returned to the problem program.

If the pseudo-page-fault was a completion, interrupt status is not saved. For completions the waiting task is found in the page wait queue and posted dispatchable. The previous completion address is set equal to the current one for the duplicate test and control is returned to the dispatcher.

If the pseudo-page-fault is an exception it is tested to see if it occurred in the dispatcher. If this is the case control is returned to the dispatcher with disabled PSW. If the fault is not in the dispatcher the page fault address is saved in the TIB, the TIB is enqueued in the page wait queue and the task is set to the WAIT state.

NAME	ID	MEANING	ACTION
SYSTEMID	00	System error condition, for example, page fault in the I/O interrupt handler	Hard wait. (see Note)
REENTRID	04	Page fault or GETREAL request in a reentrant routine	Save PSW and registers (general purpose + AR) in user task's second save areas.
USERTID	08	Page fault from a user task or from a system task.	Hard wait X'FFB' if this is a system task and the TCB shows that the task does not expect page faults; else registers (general purpose and floating point) and interrupt status are saved in the user's save area, AR in task's 1st AR save area. If the task operated in disabled mode, the task is canceled with cancel code X'15'; otherwise the page request is enqueued.
APPENDID	0C	Page fault in I/O appendage routine	Task is canceled with cancel code X'36'.
RESVCID	10	Page fault in SVC 7 (X'07') in SVC 13 (X'0D')	Set RETRY SVC bit in TIB save interrupt status and registers in user save area; enqueue page request.
DISPID	14	Page fault in a routine which does not require any information to be saved, e.g. page fault in the dispatcher.	Enqueue page request.

Figure 13 (Part 1 of 2). Routine Identifiers (RID) as Used by the Page Fault Handler (PFFLIH)



---

## External Interrupt

The external interruption provides a means by which the CPU responds to various signals originating either from within or from outside the system. The sources that may present a request for an external interrupt are:

- Clock comparator
- CPU timer
- External interrupt key
- External signal
- VM/IUCV
- APPC/VM

---

## Machine Check Interrupt

The resident machine check handler (MCH) analyzes the machine check interruption code and tests the problem state bit (Old PSW bit 15). The action taken depends on the conditions detected. For a more detailed description refer to "Machine Check, Channel Check and CRW Handling" on page 367 later in this chapter .

---

## Supervisor Call Interrupt (SVC)

The different processing routines are entered by the First Level Interrupt Handler (FLIH). Some SVCs are optional and cause a CANCEL (ERR21) if the supervisor was generated without the appropriate option.

After completion of the requested service (SVC), control is generally passed to the task selection routine. The only exception is SVC 107 (X'6B' - FASTSVC) which may return directly to the issuing program.

If the execution mode of a SVC differs from that defined in the table SVCMODT, the SVC is cancelled with 'execution mode violation', (ERR45).

Execution mode comprises

- addressing/residency mode
- access register mode
- cross memory mode

When a SVC processing routine gets control, it can find the execution mode of the caller in the flag TCB.TCBEXFLG.

Normally, a SVC processing routine will get control in AMODE 24 and switch to AMODE 31 by itself whenever necessary.

With a few exceptions, SVCs are not allowed in cross memory mode.





---

# Dispatcher

---

## z/VSE (Turbo) Dispatcher - Introduction

This paragraph gives an introduction to the Turbo Dispatcher. First we will discuss the VSE partition and VSE task concept. VSE tasks are split into system tasks, maintasks and subtasks, VSE partitions into one system partition, static and dynamic partitions.

## Comparison System Task / User Task

For better understanding of system tasks processing it is important to distinguish between server and service owner.

1. Normally a system task is performing the service which has been requested by a user task.  
In this case
  - the system task is the server and
  - the user task is the requester and owner of the service.
2. A system task may request participation of another system task on the same service.  
In this case, the service owner will remain the original service requester whereas the first system task will be the immediate service requester to the second one.
3. System tasks (for example, attention task) may perform processing which is not connected to any kind of user task.  
In this case (similar to user task) a system task is server and service owner of its own.

## System Partition

In order to allow system and user task selection by the same mechanism identical control blocks are used with both kinds of tasks. In addition to the user partitions a pseudo partition (system partition) is used, which is the home of all system tasks including attention. Task selection differentiates between two control blocks which are related to partitions. These are

- Partition Control Block (PCB)
- Partition Information Block extension (PIB2)
- Partition Communication Region (COMREG)

A PCB represents a partition as the server whereas the PIB2 and COMREG are representing the service owner. This means that in case of system task processing the system PCB is involved in a combination with a user PIB2 and COMREG whereas in the case of user task processing the PCB, PIB2 and COMREG belong to the same user partition. If a system task has no service owner, the system PCB, attention routine PIB2 and BG COMREG represent the selected environment.

## Static Partitions

Supervisor generation defines control information for 12 partitions, called **static partitions**. Static partitions are compatible to the partitions in prior releases, that is they support the old control block structure (defined by the partition COMREG (SYSIR) interface) as well as the new control block structure defined by the PCE (Partition Control block Extension). VSE allows to allocate 12 static partitions in up to 12 address spaces. An address space contains a shared area and a private area. The shared area is for all address spaces the same. The private area differs dependent on the allocation. It is possible to allocate more than one partition in one private area. Static partitions have the following predefined IDs: F1, F2, F3, F4, F5, F6, F7, F8, F9, FA, FB, BG.

The following storage layout shows partitions BG, F1, F2, F3 and F4 allocated in address spaces 0 to 3; one address space has a maximum size of 2 GigaByte (GB).

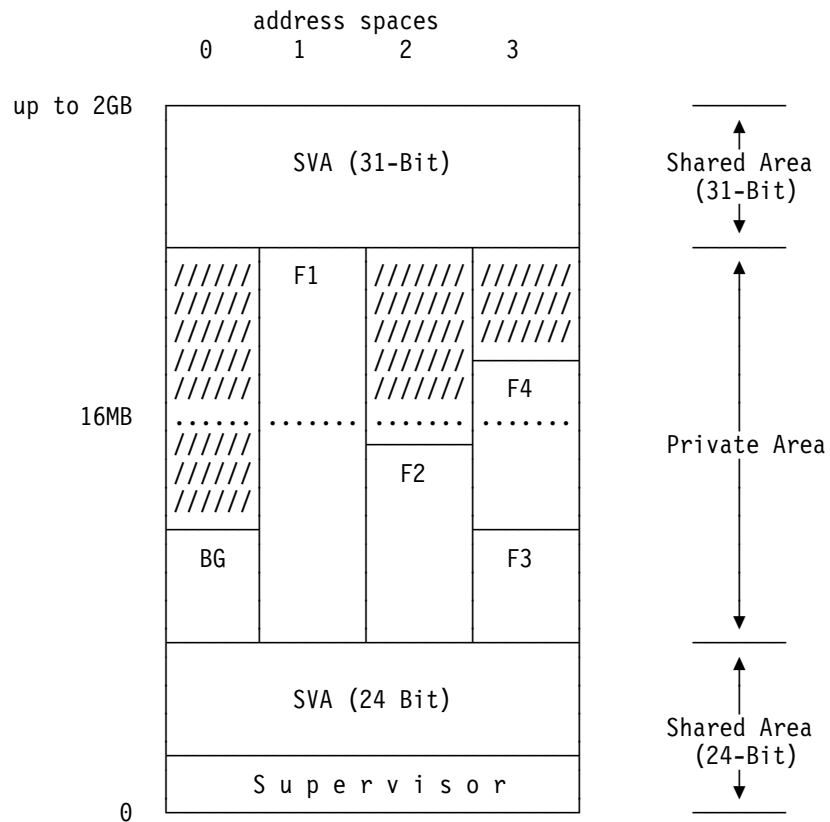


Figure 14. Storage Layout (Static Partitions)

An operator or ASI procedure may allocate and start the partitions and give an execution priority to every partition. Static partitions may get JCL statements from a physical reader or the job scheduler. In VSE the job scheduling, execution and output spooling of a job is done by VSE/POWER, that is a job is VSE/POWER controlled. VSE/POWER may also start the static partitions and run jobs in them, that is VSE/POWER feeds JCL statements and data to the partition. The jobs are located in the reader queue. The list or punch output may be directed to the corresponding queues. The static partition is available for more than one job. VSE/POWER allows to execute jobs in classes, the so called VSE/POWER classes, where one or more static partitions may be assigned to.

## Dynamic Partitions

**Dynamic partitions** (PCEDYNP in PCEFLAG) have a few incompatibilities compared to static partitions. Two byte control block interfaces accessed via the COMREG (SYSIR interface) are not supported, for example, PIBTAB, PIB2TAB, DIB, NICL, FICL, etc., because partition related control blocks are allocated in the SVA (24-bit, for example, PIB, PIB2, PCB, COMREG, SCB, TIB, TCB). That is for dynamic partitions the supervisor only supports the new control block structure defined by the PCE (Partition Control block Extension).

Also a few supervisor services are not available in a dynamic partition, for example, PFI, XECB, EXEC REAL, SYSFILE support. In the following you will find a description of the dynamic partition characteristics.

VSE/POWER allocates and starts a dynamic partition for one job. VSE/POWER controls the execution of the job and deallocates the dynamic partition after end of job. The freed storage is available for another job.

Each dynamic partition has its own address space, that is only one dynamic partition is allocated in the private area of such an address space. Multiple dynamic partitions may be allocated at a time.

To reduce the size of the shared areas a new (private) system area is introduced for dynamic partition address spaces called **dynamic space GETVIS area**. The dynamic space GETVIS area belongs to the private area. Figure 15 on page 32 shows a comparison of static and dynamic Partition address space layouts.

The dynamic space GETVIS area is allocated together with the dynamic partition and contains system data for the address space.

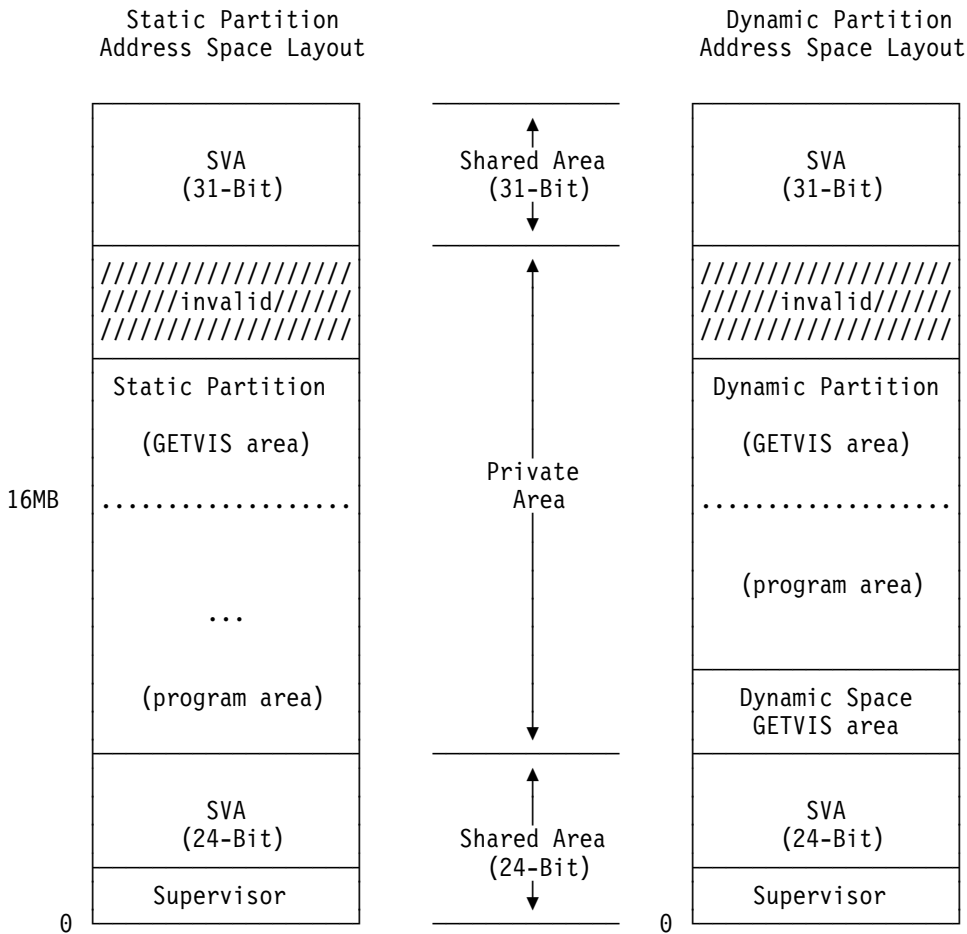


Figure 15. Address Space Layout for Static and Dynamic Partitions

Dynamic partitions are grouped into classes corresponding to the VSE/POWER classes, where jobs can be executed. These classes are called **dynamic classes**.

Only the priority of the dynamic class can be specified. The dynamic partitions within the class are time sliced.

Static and dynamic partitions may be grouped in one VSE/POWER class.

A table, the so called **dynamic class table**, contains the attributes of the dynamic classes. The dynamic class table may be cataloged in the library, from where it can be loaded.

Some dynamic class table attribute examples:

- *Storage allocation:*  
specifies the storage requirements of a dynamic partition (Dynamic Space GETVIS and partition allocation) of the given class,
- *Partition SIZE:*  
defines the amount of contiguous virtual storage in a partition reserved for program execution; the rest of the partition is available as partition GETVIS area,

- *Dynamic Space GETVIS size:*  
defines the amount of contiguous virtual storage for the Dynamic Space GETVIS area (refer to Figure 15 on page 32),
- *Profile(procedure):*  
will be executed in the partition prior to the job (corresponds to ASI procedure of static partitions),
- *Max. number of dynamic partitions within class:*  
specifies the maximum number of partitions that can be allocated in parallel within the given class, when enough virtual storage and dynamic partitions are available;
- *Disable indication:*  
allows to disable a dynamic class, that is no job can be executed within this class;
- *Spooled I/O devices:*  
specifies the spooled devices, for example, reader, printers and punches which interfaces to VSE/POWER.

Dynamic partition IDs are built as follows:

<class><pno>

where <class> = dynamic class (one character, defined by user)  
<pno> = partition number within dynamic class

**Examples**

1. The first dynamic partition allocated in a dynamic class *P* receives ID *P1*.
2. The following example shows a storage layout with partitions BG, F1 - F7 in address spaces 0 to 3, 2 dynamic partitions of dynamic class *N* in address spaces N1 and N2, one dynamic partition in address space P2 and one dynamic partition in address space 04.

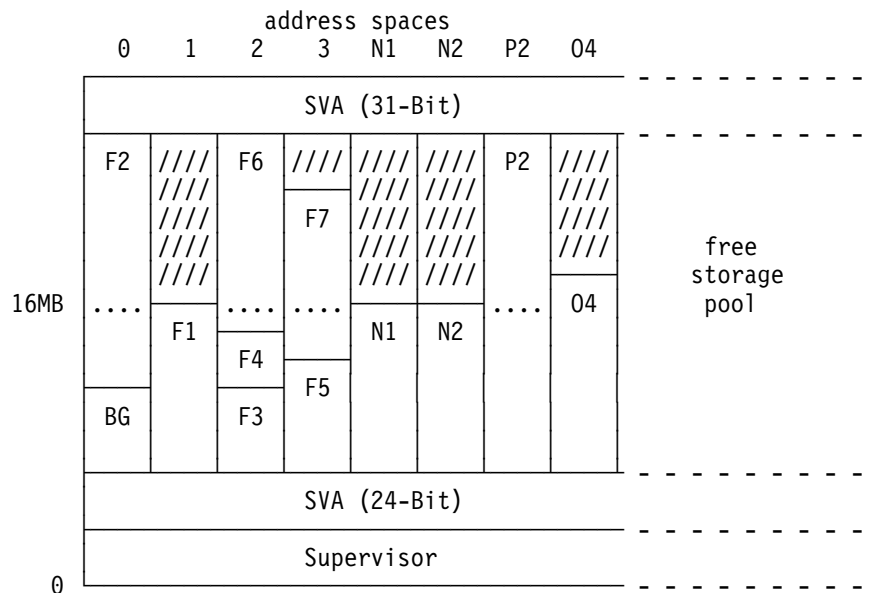


Figure 16. Storage Layout with Dynamic Partitions

## Dynamic Partitions and Ease of Use

The following items show how easy it is to process jobs in dynamic partitions:

1. Definition:  
The dynamic class table allows to define the attributes of dynamic partitions in only a few statements.
2. Dynamic partitions allocated when needed:  
Dynamic partitions are only allocated, when a job is to be executed in a dynamic class. This saves system resources. If a VSE/POWER class contains static and dynamic partitions, static partitions will be used first for job execution (default, may be changed by VSE/POWER startup parameter).
3. Automatic allocation, initialization and deactivation:  
The job's environment and resources are allocated and released automatically without user intervention.
4. Identical resource allocation within one dynamic class:  
Resources needed for the job are allocated prior to job execution.
5. Time slicing:  
There is no need to control CPU intensive jobs, because after the time slice of a job is exhausted, the job is moved to lowest priority position within the dynamic class.
6. Migration from static to dynamic partitions:  
It is easy to migrate jobs to dynamic partitions, because the design avoids incompatibilities.

## Dynamic Partition Concept

The concept shows, how dynamic classes can be defined and how dynamic partitions are created and released, when the system resources are available (for example, virtual storage).

The following paragraphs describe

1. dynamic class table maintenance,
2. dynamic partition dispatching,
3. job execution.

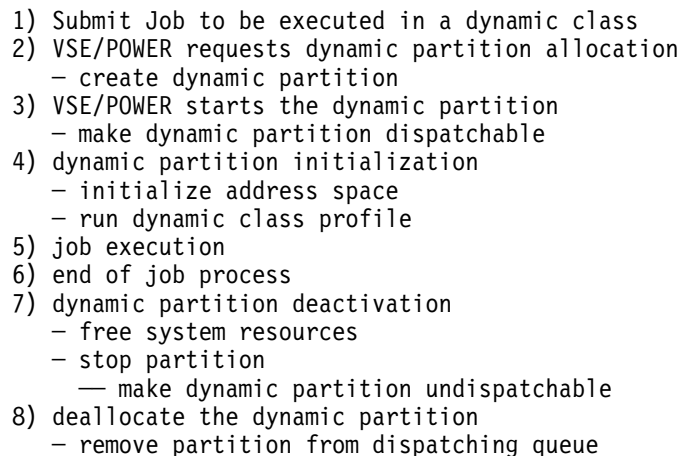
- 
- 1) Submit Job to be executed in a dynamic class
  - 2) VSE/POWER requests dynamic partition allocation
    - create dynamic partition
  - 3) VSE/POWER starts the dynamic partition
    - make dynamic partition dispatchable
  - 4) dynamic partition initialization
    - initialize address space
    - run dynamic class profile
  - 5) job execution
  - 6) end of job process
  - 7) dynamic partition deactivation
    - free system resources
    - stop partition
      - make dynamic partition undispachable
  - 8) deallocate the dynamic partition
    - remove partition from dispatching queue

Figure 17. Job Execution - Overview

## **Maintain the Dynamic Class Table (Create, Store and Load)**

Any editor or the VSE dialog may create and change the dynamic class table. Services are available to store the table into the library from where the operator or the VSE/POWER ASI procedure may load the table.

The load process validates the dynamic class table values and translates the information into an internal representation. Invalid entries are shown on the console. Only valid entries can be activated, that is the dynamic class table is stored into system storage (addressable by system routines). The dynamic classes receive a predefined dispatching priority. An operator may change the priority by the priority command.

A following load request may change or remove dynamic class table entries. A load request is rejected, when a dynamic class to be removed has allocated dynamic partitions.

## **How to Change the Priority**

The operator or a startup procedure may change the priority of static partitions and dynamic classes. To avoid that a CPU intensive partition/class can stop the dispatching of lower priority partitions/classes VSE allows to define a time sliced (balanced) group. The member of the group (static partition or dynamic class) that loses its time slice receives the lowest priority position within the group. If more than one dynamic partition is allocated within a dynamic class the same algorithm is used.

## **Job Execution**

Job selection for execution in a dynamic class is only possible, if the dynamic class is enabled. This is indicated in the loaded (active) dynamic class table and can be changed by a VSE/POWER command. This paragraph describes the processing after a job for an enabled dynamic class is found.

### ***Dynamic Partition Allocation and Start***

VSE/POWER requests allocation of a dynamic partition. The allocation service builds a partition ID, allocates and initializes control blocks and creates an address space. The dynamic partition is included into the dynamic class dispatcher queue. The partition start service makes the dynamic partition dispatchable (ready to run) and starts initialization.

If a second dynamic partition is allocated within the same dynamic class, time slicing for the dynamic class is started.

### ***Dynamic Partition Initialization***

During dynamic partition initialization the address space is validated, job control is loaded and the dynamic partition prepared. A few system routines may be executed during preparation. Job control executes the profile.

If the initialization cannot complete, the partition will be canceled, deactivated, deallocated and the dynamic class disabled to avoid execution of other jobs.

After successful initialization the job is executed, that is VSE/POWER passes JCL and data to the dynamic partition. VSE/POWER requests deactivation, when the end of the job is reached.

### ***Dynamic Partition Deactivation***

System routines called during preparation are notified that deactivation is requested. The system frees resources, stops the dynamic partition and makes the partition undispatchable. VSE/POWER deallocates the partition.

### ***Dynamic Partition Deallocation***

The deallocation service frees the partition ID and system space allocated for control information (for example, control blocks). It removes the dynamic partition from the dynamic class dispatching queue. If only one dynamic partition remains in the dynamic class, time slicing for the dynamic class will be reset.

### **Advantages of the Concept**

The customer has only to define system resources available for dynamic partitions (for example, virtual storage size, dynamic class table). The job execution including partition allocation and deallocation is done by the system, that is system resources are only used when needed. It is easy to migrate jobs from static to dynamic partitions, because the incompatibilities (control blocks, services, etc.) are kept as low as possible. Only system software should be affected. The dynamic partition concept is open for follow on development.



---

## Turbo Dispatcher

### Turbo Dispatcher Design

A z/VSE system with the Turbo Dispatcher (TD) active can run on any supported uni- or multiprocessor. The Turbo Dispatcher can utilize multiprocessors by distributing the workload across several processors (CPUs) of one Central Electronic Complex (CEC), enabling them to work in parallel and thus increase the overall throughput of a z/VSE system.

At Initial Program Load (IPL) the Turbo Dispatcher (\$IJBDSPT) phase will be loaded just after the z/VSE Supervisor.

The z/VSE Turbo Dispatcher works on a partition (job) basis, that is, it dispatches an entire partition to a CPU waiting for work, instead of dispatching at a subtask level like OS/390 does. Subsequently "jobs" is used as a synonym for partitions. One job consists of many work units. A **work unit** is defined as a set of instructions that are executed from the selection by the z/VSE Turbo Dispatcher until the next interrupt. Only one work unit of a job can be processed at a time, that is, no other work unit of the same job can run on a different CPU. This means for jobs with multitasking applications (applications with attached VSE subtasks), that no other task of the same job can execute on a different CPU, when one task of that job is already active.

There are two different work unit types:

1. **parallel work units (P)**

Most customer applications in batch as well as the online (CICS/VSE or CICS Transaction Server) environment are processed as parallel work units.

However, when an application calls a supervisor service, it has to process a non-parallel work unit in most cases.

2. **non-parallel work units (N)**

Most system services and key 0 applications (such as supervisor-, VSE/POWER and ACF/VTAM services) will be processed as non-parallel work units.

Only one CPU within the CEC may process a non-parallel work unit at a time, that is as long as this work unit type is active, no other CPU can execute a non-parallel work unit. Any other job, that wants to process a non-parallel work unit, has to wait until no other CPU processes a non-parallel work unit.

However, other CPUs may process parallel work units of other jobs.

**Notes:**

- a. Work units of the VSE/POWER maintask can be processed in parallel, if the VSE/POWER autostart statement

```
SET WORKUNIT=PA
```

is specified in the VSE/POWER startup procedure.

- b. Some vendors adapted their applications to run parallel work units even if they are executing in key zero.

The following simplified example (no interrupts are considered, all work units have the same length) should give you an impression, how the z/VSE Turbo Dispatcher

processes a given workload (see Figure 18 on page 38). Job A, B and C are ready for selection, job A has highest, job C lowest priority. Each job consists of 3 work units, e.g. job A consists of work units A1, A2, and A3. Work unit A1 has to be processed before work unit A2 (of the same job) can be selected. Work units are either parallel (P) or non-parallel (N). On a uni-processor the three jobs would need 9 process steps (3 jobs times 3 work units), the z/VSE Turbo Dispatcher would need only 5 steps on a 2-way (dyadic) CEC as shown in the example (with CPU 0 and CPU 1):

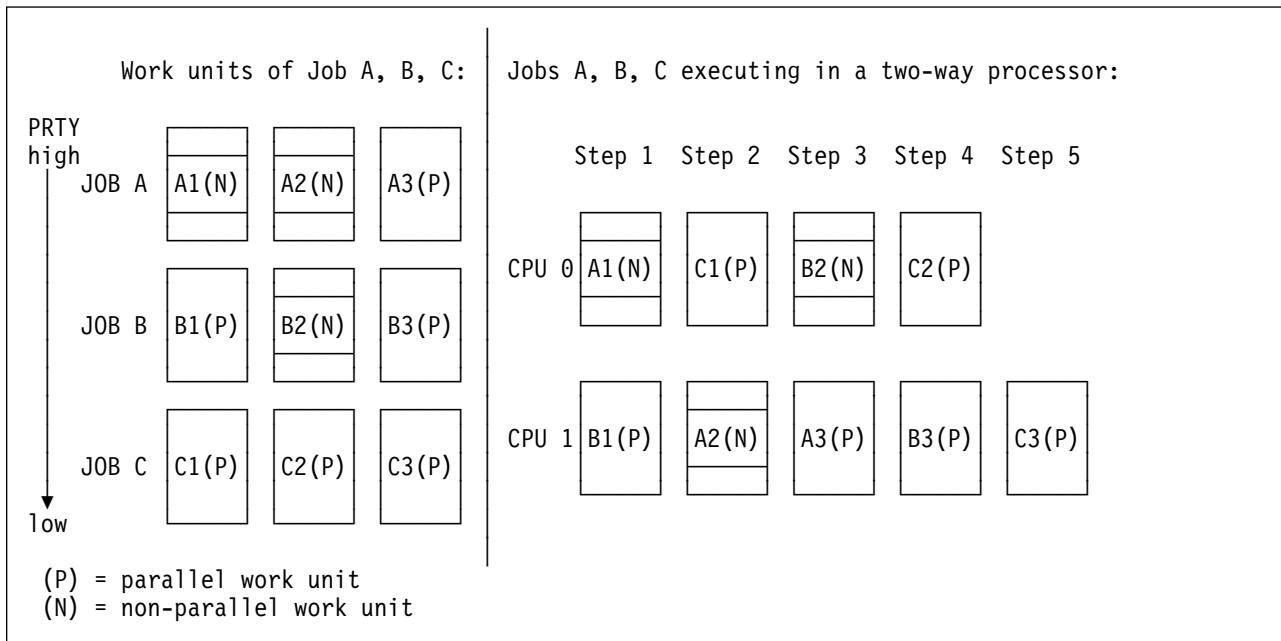


Figure 18. Processing Steps for Jobs A, B and C

- **Step 1:**  
CPU 0 selects non-parallel work unit A1, at the same time CPU 1 selects parallel work unit B1, because job A and B have a higher priority than job C.
- **Step 2:**  
The next two highest priority work units are A2 and B2. However, both work units are non-parallel and therefore cannot run at the same time. So the next lower priority parallel work unit C1 will be selected, that is this step is made of work unit A2 and C1. A2 may be selected by CPU 0 or CPU 1, in the example it executes on CPU 1.
- **Step 3:**  
Now non-parallel work unit B2 and parallel work unit A3 will be processed, as given by their priority.
- **Step 4:**  
Job A terminated. C2 and B3 are both parallel work units and will be processed by CPU 0 and CPU 1.
- **Step 5:**  
Now also job B terminated. In our example CPU 1 processes the last available work unit C3 of job C.

## Advantages on Uni-Processors

The Turbo Dispatcher provides advantages not only for multiprocessors but also for uni-processors in terms of

- multiprocessor exploitation prediction,
- measurement tools (SIR command),
- partition balancing enhancements and
- performance improvements for some environments (e.g. when VSE/ICCF is active).

## Partition Balancing Enhancements

The Turbo Dispatcher provides an improved partition balancing algorithm, which gives each partition, be it static or dynamic, equal weight within a balanced group. The PRTY command defines the balanced group, e.g.

```
PRTY F4,C=BG=F5,F3,F2,f1
```

In our example we have dynamic class C (which may hold up to 32 dynamic partitions), static partitions BG and F5 as **balanced group members**.

Balanced group members are time sliced. The calculation of a time slice size is based on the MSECs interval. The MSECs command may change this MSECs interval (default is about 1000 milliseconds).

With the Turbo Dispatcher active all dynamic and static partitions of the balanced group will receive the same time slice. That is when a dynamic partition's time slice expires, only this partition will be moved to a lower priority, all other dynamic partitions of the same class will not change their priority.

### Relative CPU Shares

The PRTY SHARE command allows to specify a **relative share** of CPU time for each static partition and dynamic class belonging to the balanced group. With this enhancement it is much easier, for example, to balance a CICS partition with batch partitions in a way that ensures acceptable throughput for the batch partitions and acceptable response times for the CICS transactions.

## Quiesce CPUs

This support was especially implemented for VSE/ESA systems running as guests under VM/ESA. Before VSE/ESA 2.3 not active (stopped) CPUs excluded the VM/ESA guest from I/O assist, which caused performance degradation. With the STOPQ parameter of the SYSDEF command it is possible to quiesce a CPU, that is a CPU will be suspended from task selection, however the CPU is still active (not in stopped state). A quiesced CPU which is not needed during a certain period of time (for example during off-shift) helps to minimize the overhead caused by idle additional CPUs.

## z/VSE Turbo Dispatcher Considerations

The z/VSE Turbo Dispatcher support is transparent to most programs as well as IBM's subsystems such as the CICS Transaction Server, CICS/VSE and ACF/VTAM. Apart from few exceptions, application programs can run functionally unchanged with the z/VSE Turbo Dispatcher. However, there may be the need to adapt applications for better multiprocessor exploitation (e.g. by implementing larger I/O buffers or using data spaces).

A few system applications run in key zero, have interfaces with the dispatcher or supervisor areas or update the first 4 KB page. These applications may have the need for adaptations. Examples for such applications are performance monitors, accounting and scheduler routines.

**Note:** The traditional replacement of the **SVC new PSW** (Program-Status Word) e.g. by vendors cause performance degradations in the multiprocessor environment. z/VSE provides vendor exits to get rid of that replacement.

Most vendor products adapted their applications to the Turbo Dispatcher environment and improved performance (compared to the standard dispatcher) by exploiting vendor exits.

Most user applications are written in high level languages (such as COBOL) and do not access internal system areas.

## More Information

You will find further details about the Turbo Dispatcher in the following publications:

- *VSE/ESA Turbo Dispatcher Guide and Reference, VSE/ESA 2.4.0,*
- *ITSO VSE/ESA 2.1 Turbo Dispatcher, SG24-4674,*
- *Hints and Tips for VSE/ESA,* available on Internet (<http://www.ibm.com/vse/>),
- *z/VSE Turbo Dispatcher Performance* document, available on Internet (<http://www.ibm.com/vse/>).

## z/VSE Turbo Dispatcher - Details

In z/VSE each CPU shares real memory and has also access to the shared areas (supervisor, SVA, etc.). The first page, call **prefix page**, is an exception to that rule, it is unique to each CPU. In z/VSE the prefix pages are allocated in virtual storage accessible by all CPUs. Each CPU has its own work area, which is also located in shared virtual storage. The supervisor code is available for all CPUs, that is the supervisor services, that are not reentrant, have to lock code.

The following figure shows the virtual storage accessible by two CPUs, where the CICS partition is assigned to CPU 0 and the VTAM partition to CPU 1 at one instant.

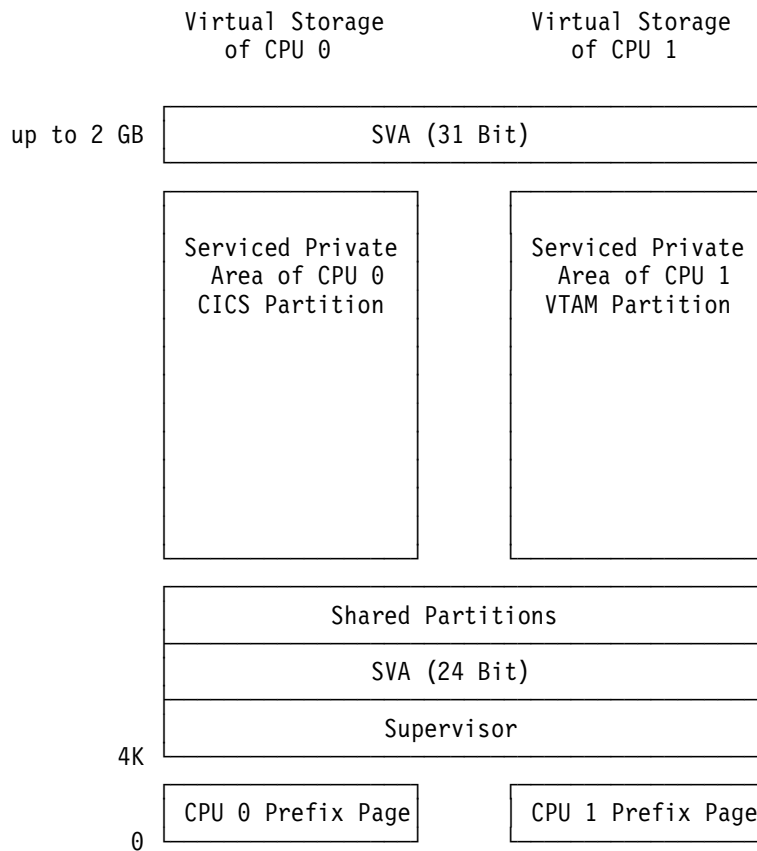


Figure 19. Sharing of Virtual Storage for one CEC (e.g. 2 CPUs)

IPL is processed on a single CPU. The z/VSE operator or VSE/AF ASI procedure may start other CPUs via a new command. The supervisor use the **Signal Processor (SIGP)** instruction to initialize and start a different CPU and to communicate in between CPUs, which is used in rare cases. In general defined interfaces located in the shared area are used for communication.

One VSE/POWER job consists of multiple work units. One work unit is defined as the dispatchable entity of the partition, which can be processed on one CPU until the next (SVC, I/O, external, etc.) interrupt.

All CPUs within one CEC have equivalent rights as long as no work unit is assigned to a specific CPU. One job consists of parallel and non-parallel work units.

**Parallel work units (P)** are defined as application code like CICS transactions (running in problem state and non-key zero). **Non-parallel work units (N)** are defined as system code like supervisor services, VSE/POWER, or ACF/VTAM (key 0, disabled and/or supervisor state work units).

Whenever (N) code is executing on one CPU (lets say CPU X), all other CPUs that request execution of (N) code have to delay these work units or wait for the (N) state, that is no other (N) code can run on a different CPU when active on CPU X. However, any other CPU may process (P) code. When the (N) work unit has to be delayed, a CPU may select another (P) work unit of a different partition, if available. After CPU X has processed a (N) work unit, it looks for other enqueued (N) work units. If no such (N) work units are available, any other CPU may run a (N) work unit; CPU X will select the highest priority (N) or (P) work unit, if any.

**Examples:** The following examples show the assignments of two partitions to two CPUs (CPU 0, CPU1), all ready-to-run, where the following abbreviations are used:

- P1 = Partition 1
- P2 = Partition 2
- (P) = executing parallel work units
- (N) = executing non-parallel work units

The examples also show the transition from (P) work units to (N) work units.

*Example 1: Supervisor Call (SVC) Interrupt Process:* The first example uses SVCs to switch from (P) work units to (N) work units. The z/VSE TD decides, if the CPU has to wait (using spin loops) until it can process the (N) work unit for the corresponding job (partition) or delay the SVC and select another ready-to-run job.

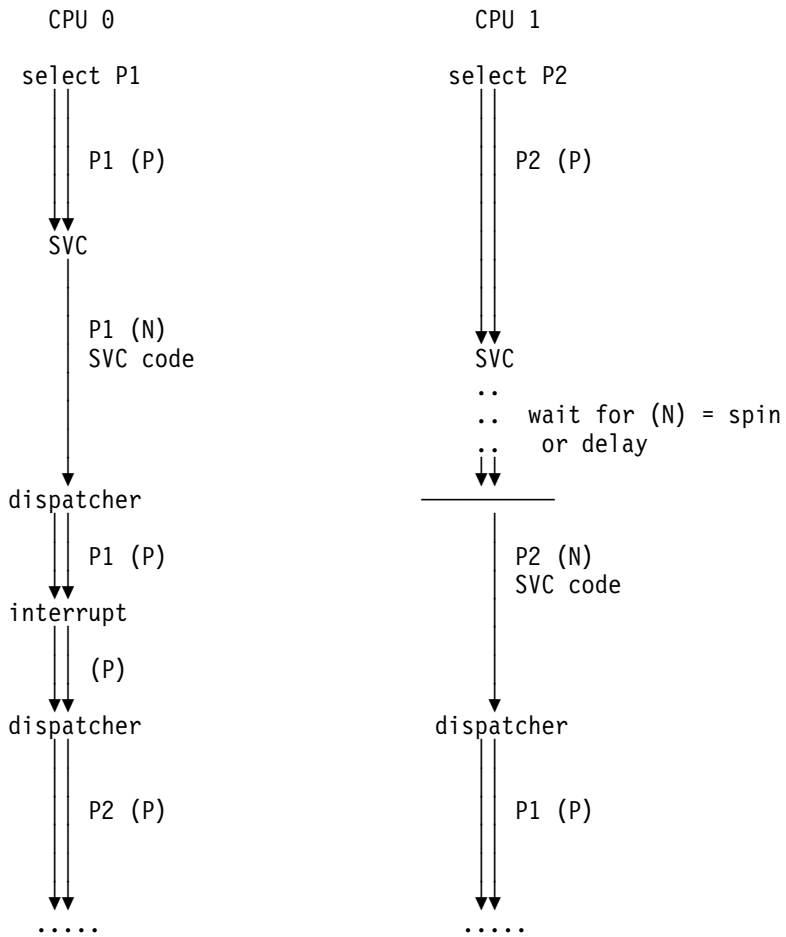


Figure 20. SVC Interrupt Process

*Example 2: Clock Comparator and I/O Interrupt Process:* The second example shows how I/O and clock comparator (external) interrupts are processed. The CPU that receives an I/O or external interrupt tries to get the N state. If this is not possible, it enqueues the interrupt to the CPU that runs (N) work units and continues with interrupted (P) work unit (partition) or frees the work unit and continues with the next higher priority work unit.

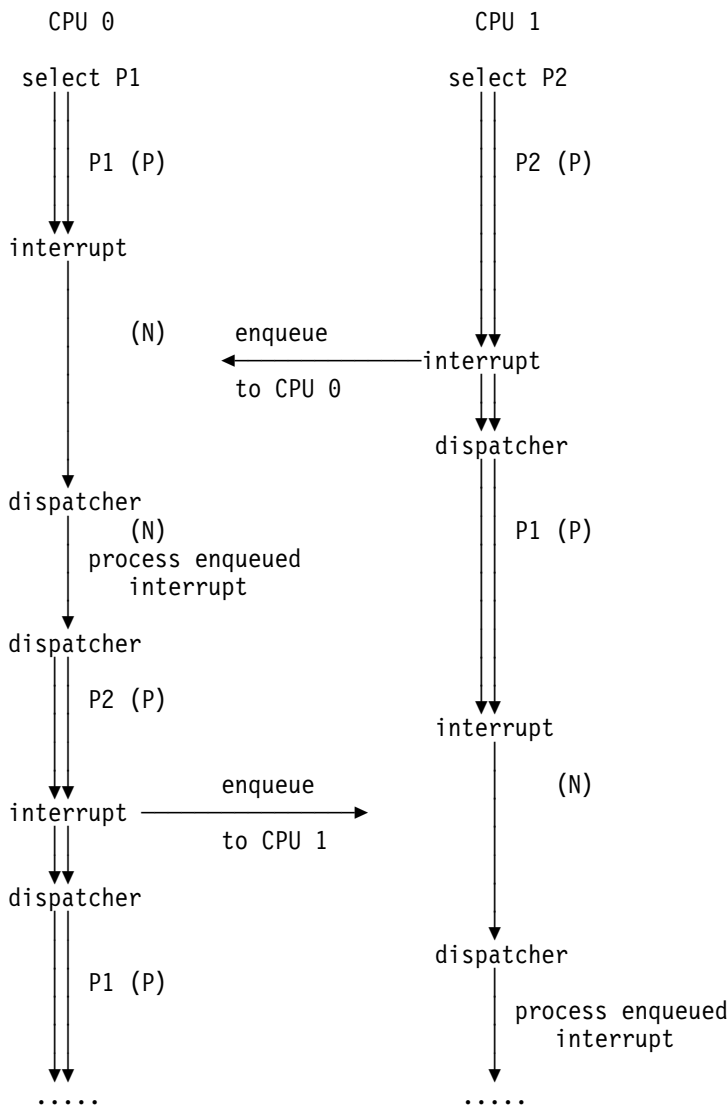


Figure 21. External and I/O Interrupt Process

Parallel work units are enabled for I/O and external interrupts like in prior VSE/ESA releases. So any CPU may receive interrupts.

**Note:** With a few exceptions, e.g. IUCV and VMCF interrupts are enabled only on that CPU, where IPL was performed.

Work units are selected dependent on the z/VSE priority scheme.

**Note:** The z/VSE Turbo Dispatcher will be implemented in separate module, which is OCO (Object Code Only). Therefore you will find more details about the Turbo Dispatcher in an internal document.



## **z/VSE Turbo Dispatcher - CPU Management**

The SYSDEF AR and JCL command allows to start and terminate all available or selected CPUs.

The following paragraphs describe

- Activation and initialization of CPUs (Multiprocessing)
- Termination of CPUs (Multiprocessing)
- CPU Recovery

### **Activation and Initialization of CPUs**

A new AR/JCL SYSDEF operand will be introduced to activate z/VSE multiprocessing capabilities. All or one additional CPU can be activated.

The initialization process determines the number of CPUs available in the CEC configuration (no supervisor generation is required) and allocates prefix pages (page 0) and work areas for each CPU in system GETVIS (31 bit) storage. SIGP instructions are used to initialize and start the CPUs. A CPU vector table (TDTAB) will be allocated in virtual storage (outside page 0). The vector table holds an address for each CPU pointing to the CPU control block (TCPU) of the corresponding CPU. TCPU points to the CPU's work area. Each CPU has access to the other CPUs page 0 and work areas.

### **Termination of Multiprocessing**

A new AR/JCL SYSDEF operand will be introduced to terminate z/VSE multiprocessing capabilities. One specific CPU or all active CPUs are stopped using SIGP instructions. When all additional CPUs are to be stopped, z/VSE continues to run in uniprocessor mode on the 'IPLed' CPU.

## **z/VSE Turbo Dispatcher - Data Areas, Control Blocks and Structure**

### **System Areas (Control Blocks)**

Updates to system control blocks are only possible by tasks running with key zero, which means in z/VSE's TD approach, can be changed from one CPU at a time. Control blocks and areas that are changed dependent on the status of the system should be removed from page 0 (low core).

The following control blocks and areas will be moved from page 0:

- Resource descriptor table
- TIBATAB (Task Information Block Address Table)  
will be located at the 4K boundary, the static part will stay in page 0, the remaining part in page 1.
- RASLINK area
- whenever possible: move areas out of page 0

Most critical control blocks that are committed to applications are the SYSCOM and the BG COMREG, because they are allocated in the page 0. These control block will remain in page 0. When the task, running a non-parallel work unit, updates these control blocks, the update will become active on the other CPUs, when the (N) work unit is completed ((N) state is freed - in dispatcher or by service).

It may also be necessary to lock critical sections of code across all CPUs.

**Note:** Locking facilities will be provided to allow synchronization of control block updates.

## CPU Control Blocks

This Paragraph gives an overview of the CPU control blocks in z/VSE:

- TD Address Table (TDATAB)
- TD Communication Region (TDCOMREG)
- TD CPU Control Block (TCPU)

**TD Address Table (TDATAB):** The TDATAB holds addresses to all z/VSE CPU control blocks. The first entry points to the TDCOMREG, the following up to 10 entries point to the TCPU control blocks of the corresponding activated CPUs. The first entry with the high order byte set gives the first unused TDATAB entry.

The SYSCOM holds the address of the TDATAB at label IJBTDATB.

**TD Communication Region (TDCOMREG):** The TDCOMREG holds system information, e.g.

- started CPUs
- control information
- delayed I/O and external interrupts
- global spin (compare-and-swap) locks

The TDCOMREG is allocated in the supervisor.

**TD CPU Control Block (TCPU):** The TCPU holds information specific for the corresponding CPU: such as:

- CPU id
- active work unit type ((N) or (P))
- status of CPU
- process requests like
  - process PALB
  - reset CPU time counters
  - set default space
- CPU's PREFIX page (virtual and real)
- CPU's work area address
- accounting information
  - total CPU time since last IPL or reset
  - total (N) time since last IPL or reset
  - total spin time since last IPL or reset
  - total allbound time since last IPL or reset
  - dispatcher cycles since last IPL or reset

The TCPU for the IPLed CPU is allocated in the supervisor. The TCPU's for the other CPUs are allocated in system GETVIS (31 bit) storage, when active.

## Task Selection Control Blocks (TSCx)

**Note:** Task Selection Control blocks (TSC) are described in more detail in an internal document.

The TSC holds the control information for a partition or task and is used by dispatcher services and task (work unit) selection. Dynamic Class related control infor-

mation is located in the **TSCC**. Partition related control information is located in the **TSCP**. Task related control information is located in the **TSCT**.

Information located in TSCC:

- dynamic class character
- CPCB address
- partition status (active on CPU, ready-to-run)
- the priority chain forward and backward pointers
- priority figures

The TSCC which is part of the priority chain will be replaced by a dynamic partition TSCP, when the corresponding dynamic partition is allocated.

Information located in TSCP:

- partition identification key (PIK)
- PCB address
- partition status (active on CPU, ready-to-run)
- address of ready-to-run TSCP with next lower priority
- the priority chain forward and backward pointers
- address of highest priority ready-to-run TSCT
- priority figures
- address of TSCT active on any CPU
- page fault device information

The Partition Control Block (PCB) holds the address of the corresponding TSCP. The system task and BG TSCPs are located in the supervisor, all other TSCPs are allocated in system GETVIS (31 bit) storage.

Information located in TSCT:

- task identification (TID)
- TIB address
- TSCP address
- task status (active on CPU, ready-to-run)
- address of ready-to-run TSCT with next lower priority
- the priority chain forward and backward pointers
- priority figures
- address of TCPU, where task is active

The Task Information Block (TIB) holds the address of the corresponding TSCT. The system task and BG maintask TSCTs are located in the Turbo Dispatcher phase, all other TSCTs are allocated in system GETVIS (31 bit) storage, when possible.

## **z/VSE Turbo Dispatcher - Task Selection**

The z/VSE Turbo Dispatcher introduced a new task selection algorithm working on queues of work units. One work unit is described by a task selection control block, **TSC**. The queue elements (TSCs) are chained in priority order via forward pointer and backward pointers. Each queue element holds a priority number (highest number has highest priority).

VSE/AF has two TSC states:

1. TSC elements for non-parallel work units - TSC(N)

TSC(N) elements can be processed by only one CPU at a time, no other TSC(N) element can be selected.

2. TSC elements describing parallel work units - TSC(P)

TSC(P) elements can be processed by any CPU, other CPUs may select remaining ready-to-run TSC(P) elements.

VSE/AF has three TSC types:

1. the dynamic class TSC (TSCC), which is also part of the partition queue as long as no dynamic partition within the dynamic class is active,
2. the partition TSC (TSCP), which is part of the partition queue, and
3. the task TSC (TSCT), which is part of the task queue.

TSCs of Ready-to-run partitions form the **partition ready-to-run queue** and TSCs of ready-to-run tasks within a specific partition the **task ready-to-run queue**.

The system TSCP holds the anchor for the partition ready-to-run queue.

The TSCP holds the anchor for the task ready-to-run queue, which consists of ready-to-run sub- or main-tasks.

### Partition Selection

Partition/task selection will be coded reentrant, that is any CPU may scan the TSC ready-to-run queue to find a selectable element. When a CPU selects a TSCP, the TSCP will be locked and is no longer available for other CPUs.

A VSE/AF partition is uniquely assigned to a TSCP.

When no task within the partition is ready-to-run, the corresponding TSCP will be removed from the queue or is marked for deletion. The system TSCP (pointing to the system PCB) will always be part of the TSCP ready-to-run queue, it is also called the ready-to-run queue header.

If a partition with a newly readied task is not yet on the ready-to-run queue, it will be added. To ensure that partitions are dispatched in priority order and in the order in which the partitions became ready (in case of equal priority partitions - see partition balancing), the dispatcher will add a partition to the ready-to-run queue after all partitions with equal or greater priority.

### Task Selection (TSCT Queue)

Because tasks of a partition are not allowed to run concurrently on different CPUs, the TSCT queue will only be locked on the partition TSCP. A ready-to-run task with the highest priority will be selected by scanning the TSCT ready-to-run chain. A new TSCT will be enqueued into the TSCT ready-to-run queue during the ATTACH or post processing and deleted from the queue during unpost or DETACH process. An application may change the priority of a subtask by issuing the CHAP macro. Subtasks will always have a higher priority as the maintask, except the ICCF pseudo partitions. They will have always a lower priority as the maintask. The current priority of the task is formed by the partition's priority and the task priority.

When a ready-to-run task is selected, task selection process continues as described in "Relating Control Blocks to Tasks" on page 51 and "Processing of Task Selection Exit Routines" on page 52.

The dispatcher services, system task as well as the balancing routine will be implemented in an RMODE ANY module, which is located behind the supervisor phase.

### **System Resource Owners**

System resource owners are known as End-Of-Task, Terminator or LTA owners. The tasks that occupies one of these resources will receive the highest user task priority, that is the corresponding TSCT will be enqueued just behind the system TSCP as long as the resource is occupied and the TSCT is ready-to-run. All other tasks of the corresponding TSCP will run with their defined priority. When a resource owner TSCT is selected, no other task of the corresponding partition can run concurrently.

### **ICCF Pseudo Partitions**

TSCTs of ICCF pseudo partitions will always be queued to a lower priority as the CICS maintask TSCT.

## Steps to Task Selection

One VSE partition consists of one maintask and attached subtasks, where the maintask has always the lowest priority within the partition. There is one exception: ICCF pseudo (interactive) partitions have a lower priority as the maintask.

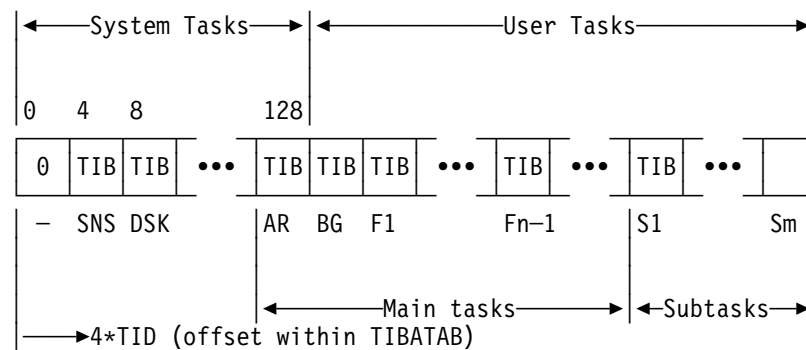
The current VSE system has a two level dispatching algorithm.

1. The highest priority partition ready to run is selected from the dispatcher queue.
2. The corresponding highest priority (sub)task ready to run is selected and dispatched. The task id (TID) is used to identify the control block structure.

### How to Identify the Control Block Structure for a given Task ?

With a given task id (TID) the task's TIB pointer can be found via the TIB address table (TIBATAB), the layout of which is shown in Figure 22.

TIBATAB



Where:

- n = number of partitions (static and dynamic)
- m = number of subtasks
- TIB = Address of TIB

Figure 22. TIB Address Table (TIBATAB)

Once a TIB pointer is known, all related control blocks and areas can be accessed as shown in Figure 23 on page 51.

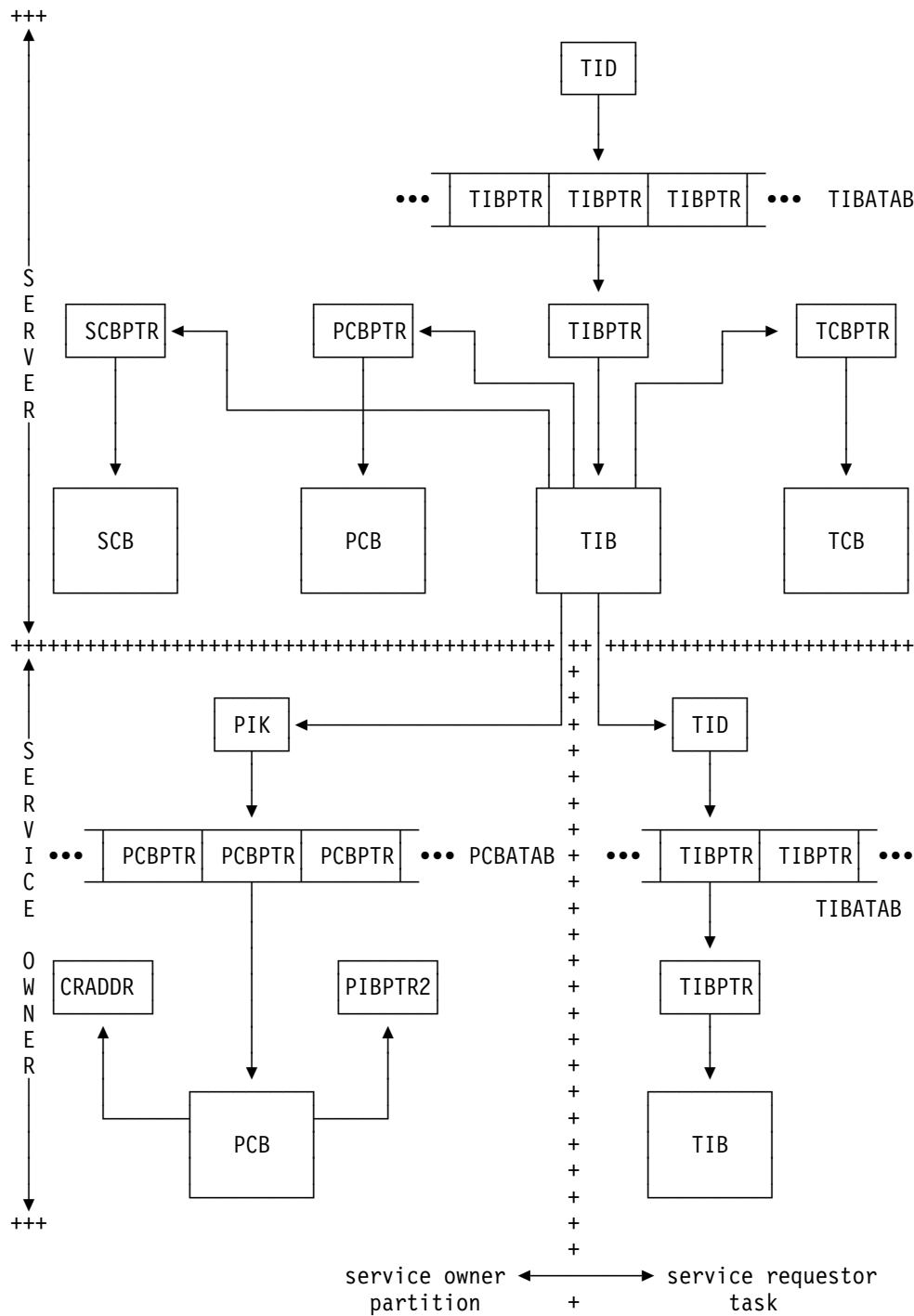


Figure 23. Task Selection Control Block Interrelationship

### Relating Control Blocks to Tasks

Control block connection is done by setting up the TID, PIK, TIBPTR, TCBPTR, PIBPTR2, PCBPTR, and CRADDR fields in such a way that they correspond to a task which has to be made active. Figure 23 shows that a control block connection can be done by assuming a given task identifier. In case of task selection, some pointers (for example, PCB pointer) are already known as a result of the first three steps of task selection.

In case of user task processing the service owner and the server are the same, that is the partition's COMREG, the PCB, the PIB2 and the PIK (CRADDR, PCBPTR, PIBPTR2 and PIK in low core) belong to the same partition. A system task may have a service owner, then the service owner's COMREG, PIB2 and PIK, but the server's PCB is used to set up the low core fields.

Once the control block connections have been established a task is active. An address space switch may be initiated by loading control register one and 7 dependent on the addressability scope of the selected task (TIBSCB in TIB). Corresponding control registers must be set in addition to support access register mode. The task's linkage stack entry address will be loaded into control register 15. But prior to returning to task processing it might be necessary to perform some supervisor services for these tasks. This is done by step 6 of task selection.

### Processing of Task Selection Exit Routines

Before a user task is activated the task selection routine tests whether control has to be transferred to any task selection exit routine.

Bits 0 to 7 of the TIBFLAG byte are associated with specific routines. They are scanned left to right and, if the bit is set to one, the corresponding routines are entered. After entry to a routine the corresponding bit is reset to zero.

There are the following exit routines:

- SVRETURN (Bit 0: X'80' - CSVRET in TIBFLAG)  
Return to an interrupted (reentrant or gated) supervisor service routine. When partition balancing and/or job accounting support is active and the new accounting owner is not the old one the current accounting interval is determined and added to the old owners time counter field (system overhead or user CPU time) and a new accounting interval is initialized. The routine identifier is moved from the TCB into the RID field.  
  
In case of a gated routine the resource (which is given by the RID) is freed and any waiting tasks are posted. The general registers of the interrupted routine are loaded from the task's system save area and control is returned to the routine loading its program status word.
- REENTSV (Bit 1: X'40' - RETRY SVC in TIBFLAG)  
Reenters the SVC first level interrupt handler routine without issuing an SVC. It is used for performance purposes. It allows a short path when the entry to an SVC routine should be retried.
- DELMOVE (Bit 2: X'20' - TIBDELMV in TIBFLAG)  
Enters the general delayed move routine. Bits 0 to 7 of the TIBDMFLG byte are associated with the delayed move routines. The routine address is determined via a left to right scan of TIBDMFLG.

One of the following routines will be activated:

- MOVECCB (Bit 0: X'80' - TIBCMVEX in TIBDMFLG)  
This exit routine has two different functions:
  1. Move a CCB which could not be copied back after completion of channel program translation because the page containing the virtual CCB was not in processor storage. Return to task selection entry.
  2. Return to SVC 119 (X'77') processing after the FBA I/O operation has been completed.
- XPCCEXIT (Bit 1: X'40' - TIBXPCEX in TIBDMFLG)



If a XPCC request is executed, where the destination is not in the same space than the originator, the control information to be stored into destination XPCCB (such as traffic bits, user data, etc.) will be saved into a supervisor control block (CRCB) and transferred to the destination XPCCB, if the associated path is dispatched.

- SV103RET (Bit 2: X'20' - TIBSFLEX in TIBDMFLG)  
If I/O is made by the SVC 103 routine, the SV103RET flag will be set in order to return to the SVC 103 routine after I/O processing.
  - TINFMOPD (Bit 3: X'10' - TIBPERST in TIBDMFLG)  
Modifies the PER active indication in the partition control block (PCB) and the save area PSW of the specified partition.
  - DISPEXRI (Bit 3: X'08' - TIBGENEX in TIBDMFLG)  
Activates the general dispatcher exit.
  - TERMSRES (Bit 3: X'04' - TIBTERMx in TIBDMFLG)  
Resets the task owning the message writer routine to its original priority, if the task is running as 'system resource owner'.
  - DISPTSTP (Bit 3: X'04' - TIBSTOPR in TIBDMFLG)  
Stops the current task (task waiting to be restarted), which was requested by the TDSERV FUNC=TASKSTOP service.
- CNCLEXIT (Bit 3: X'10' - FETCHEOJ in TIBFLAG)  
There is no save area available to be used by the resident part of the terminator routines. This exit is used to activate the terminator and to return control to it after an interruption.
  - ICCFEXIT (Bit 4: X'08' - ROLLOUT in TIBFLAG)  
It supports synchronization between an ICCF 'Pseudo Partition' task and the ICCF High Priority Task.
  - EXTRETRN (Bit 5: X'04' - CDELEX in TIBFLAG)  
This activates the user timer exit routine or posts the timer ECB after a timer interrupt for this task. Since timer interrupts are asynchronous to user task processing, activation and posting is delayed in order to have the system save area available. This is necessary because a page fault may occur when accessing the save areas or the timer ECB.
  - OCEXIT (Bit 6: X'02' - OCPEND in TIBFLAG)  
Provides delayed activation of a user OC exit routine. This is necessary because an MSG command is asynchronous to the corresponding maintask processing and the save areas involved may be paged-out.
  - APSEXIT (Bit 7: X'01' - APSEXFLG in TIBFLAG)  
Gives control to the ACF/VTAM dispatcher appendage routines (APS SWAP or ISTAPCKU routine). After returning from an appendage routine a test is made whether any OC or timer interrupts are unprocessed yet. If so, the corresponding TIBFLAG bit is set. In addition to this CNCLEXIT may be reactivated when the APSEXIT was called during EOJ processing. After processing, the APSEXIT routine returns to the entry of the task selection routine.

### **Initialize Task's Processing and Give Control to it**

Before control is given to a task a test is made whether tasks program status word (PSW) is in a disabled state. If so, an interrupt window is opened, allowing for any pending interrupt to occur. The interrupt window is closed immediately. This interrupt window prevents any task from running fully disabled (that means over a boundary of supervisor services). When partition balancing and/or job accounting support is active and the new accounting owner is not the old one the current

accounting interval is determined and added to the old owners time counter field (system overhead or user CPU time) and a new accounting interval is initialized. For a maintask which is task timer owner the remaining time slice is set. At the end of task selection the Routine Identifier (RID) field is set to the value USERTID. This indicates that normal tasks processing is active. The task's floating point, access registers and general registers are loaded and control is given to the task loading its Program Status Word (PSW).

---

## VSE/AF Dispatcher - Internal Gating Mechanism

The internal gating mechanism controls the usage of internal resources.

Its function is to

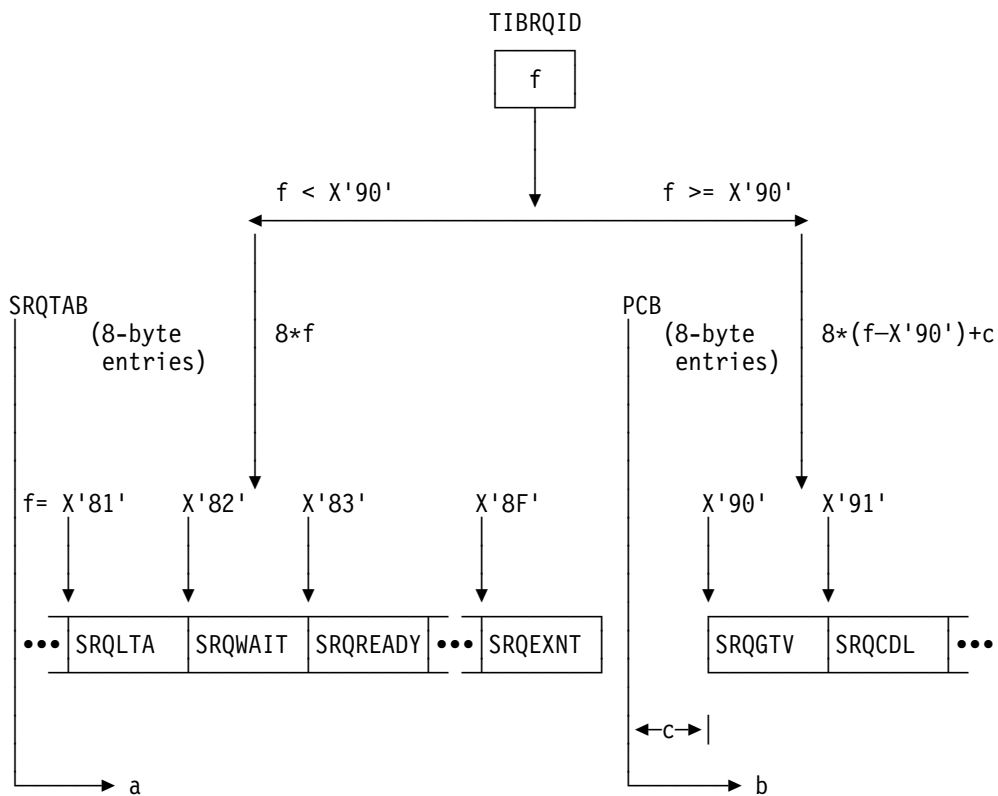
- Post/unpost Tasks and Partitions
- Free/occupy Resources
- Maintain Wait Queues

Flags, fields, tables involved in internal resource handling are:

- Partition and Task Selection String (PSS, TSS)
- Task Status Flags (located in TIB, label TIBRQID in the supervisor - Figure 26 on page 59)
- Resource descriptors (located in SRQTAB and in PCBs) including a header for building wait queues
- Wait Queues (chains of TIBs enqueued on a resource)

For the Turbo Dispatcher all ready-to-run partitions and tasks have to be on the TSCP/TSCT ready to run queues. A more exact description of a task's status is given by its task status byte (TIBRQID) and the corresponding resource descriptor.

## Addressing Resource Descriptors



- f = Value in TIBRQID byte (task status flag)
- c = Displacement of first descriptor (SRQGTV) within PCB
- a =  $8 * f$  (displacement to an entry in SRQTAB)
- b =  $8 * (f - X'90') + c$  (displacement to an entry in PCB)

Figure 24. Addressing Resource Descriptors

## Resource Descriptors

For compatibility and performance reasons there are different gating concepts implemented. The method which has to be used with a given resource is specified via a resource descriptor entry, shown in Figure 25.

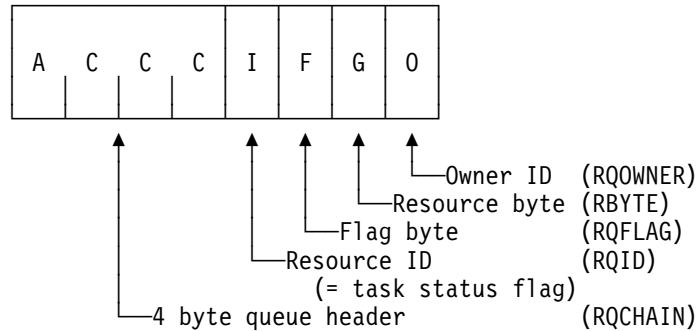


Figure 25. Resource Descriptor Entry

### Description of Entries

- ACCC: Queue header

In combination with specific resources the queue header is used for building wait queues.

- A = X'80' (first byte of a queue pointer) indicates end of a wait queue. In this case pointer ACCC points to the first byte of the corresponding resource descriptor.
- A = X'00' (first byte of a queue pointer) indicates a (or another) waiter is enqueued on a resource. In this case pointer ACCC points to the first byte of the waiters TIB.

A = X'80' in a queue header indicates that there are no waiters enqueued on the resource. A task is enqueued on a resource inserting its TIB to the front of a wait queue.

**Note:** The symbolic names of gates, their types and displacements (flag values) are shown in Figure 26 on page 59.

- I: Resource ID:  
For identification purposes, byte 4 of each entry contains the corresponding task status flag value. For example, in the entry SRQREADY, I = X'83'.
- F: Flag byte (Resource Queue ID):  
specifies the gating method to be used.

Flag	Labels	Apprev.	Type Description
X'80'	SYSTQ	S	system queue, priority posting, switchable gate
X'40'	PARTQ	P	partition queue, priority posting, switchable gate
X'20'	WAITCHN	T	TIB chain, selective posting, permanently closed gate
X'10'	IOCHN	I	I/O chain, selective or direct posting, permanently closed gate
X'08'	PGATE	C	no queue, direct posting, permanently closed gate
X'04'	PREADY	O	ready to run state, permanently opened gate
X'01'	NORDY	N	do not ready task for cancel

- **G: Resource byte (Gate):**  
The most significant element of internal resource handling is a resource byte, known as a gate. The content of the resource byte is used as a switch:
  - G = X'00' : a resource is occupied (NOTFREE)
  - G = X'80' : a resource is free (FREE)
  1. *Switchable gates: (P or S)*  
The content of a switchable gate may be changed. It may represent a single item resource (routine, system task, etc.) or multiple items of a resource (channel queue, copy buffers, etc.). Services are provided to close/open the gate, dequeue/enqueue waiters.
  2. *Permanently opened gates: (O)*  
They are used in combination with the ready to run status of tasks. Whenever a task is ready to run its TSS bit is turned on and its status flag points to a permanently opened gate.
  3. *Permanently closed gates: (C, I or T)*  
They are used in combination with the not ready to run status of tasks when switchable gates cannot be used. They are assigned to fixed owners. Tasks pointing to permanent gates are posted/unposted individually by the resource owners upon completion of a service (I/O, program fetch, etc.).
- **O: Owner ID:**  
ID of resource owner (Task ID).

## Task Status Flags

Type	Value	Name	Usage
S	47	XMSBND	Gate for cross memory services
S	48	DOCABND	Gate for router automatic storage
S	49	DOCBBND	Gate for router buffer space
S	4A	G122BND	Gate for SYSDEF SVC
S	4B	PMSBND	Gate for SPMRSERV services
S	4C	PXBHIBND	Gate for TFREE in PFIX 31-bit area
S	4D	PXBLOBND	Gate for TFREE in PFIX 24-bit area
S	4E	DYNCBND	Gate for dyn. class services
I	4F	SPDTBND	Gate for SPDT task
S	50	DSPBND	Gate for data space support
S	51	TDWBND	Gate for TD
S	52		Reserved
S	53		Reserved
S	54	SV3BND	Gate for SVC 3 to wait on SYSIO
W	55	PSPFBND	Gate for Pseudo page fault processing
S	56	SPFIXBND	Gate for PFIX in SVA processing
W	57	PWSRVBND	Gate for usage of POWER service
S	58	GQMBND	Gate for usage of LOG queue manager
S	59	UNUSBND	Gate reserved for future use
S	5A	NPGRBND	Gate for usage of LUB allocation services
S	5B	VIOBND	Gate for virtual I/O support
O	5C	CONDRDY	Flag for conditional ready state
S	5D	IUCVBND	Gate for IUCV support for VCNA
S	5E	G108BND	Gate for usage of SVC-6C
S	5F	SATBND	Gate for usage of stored assign.table
S	60		Reserved
S	61		Reserved
S	62	ERQBND	Gate for error queue entry
S	63	G133BND	Gate for XPCC processing
S	64		reserved
S	65		reserved
S	66	EOTBND	Gate for EOT routine

Type: 0 = permanently opened gate  
 C = permanently closed gate  
 I = I/O chain with permanently closed gate  
 W = wait chain with permanently closed gate  
 P = partition chain with switchable gate,  
 P gates located in Partition Control Block (PCB)  
 S = system chain with switchable gate

Figure 26 (Part 1 of 3). Task Status Flags and Resource Gates

Type	Value	Name	Usage
S	67	DOCQBND	Gate for router queue access
C	68	LCKBND	Gate for LOCK file I/O
C	69	PGFXBND	Gate for page to be freed
S	6A	GSMBND	Gate for ALLOCATE processing
S	6B	THTABBND	Gate for track hold table
C	6C	SFILBND	Gate for SYSFIL I/O
S	6D	SGTVSBND	Gate for GETVIS SVA
S	6E	LQBND	Gate for security logger queue
S	6F		reserved
C	70	MICRBND	Gate for MICR I/O
S	71	GETRBND	Gate for GETREAL processing
S	72	FDIRBND	Gate for program fetch directory
S	73	SEIZEBND	Gate for SEIZE to be freed
S	74	CILBND	Gate for CIL update
S	75	BUFBND	Gate for copy blocks
C	76	ICCFBND	Gate for ICCF high priority task
S	77	PFRBND	Gate for page frames
S	78	PFGBND	Gate for page frames (occupied by TFIX)
S	79	CHQBND	Gate for channel queue entry
S	7A	DIBBND	Gate for DIB access
S	7B	CCWBND	Gate for CCW translation
W	7C	TRKBND	Gate for track to be freed
W	7D	AVRBND	Gate for AVR processing
S	7E	G41BND	Gate for ENQ/DEQ processing
S	7F	G92BND	Gate for XECB processing
C	80	NOTACT	Flag for inactive tasks
C	80	SYSBND	Flag for inactive system tasks
S	81	LTABND	Gate for LTA use
I	82	WAITBND	Gate for ECB/XECB (I/O or TIMER or POST)
O	83	READY	Flag for ready to run state
S	84	IDRABND	Gate for program fetch IDRA (old gate)
S	84	FPGMBND	Gate for program fetch IDRA (new gate)
C	85	FETCBND	Gate for program fetch processing
W	86	PGIOBND	Gate for page I/O
C	87	PMRBND	Gate for page fault processing
I	88	ENQBND	Gate for RCB to be freed
S	89	TERMBND	Gate for terminator processing
C	8A	PGINBND	Gate for page-in

Type: 0 = permanently opened gate  
C = permanently closed gate  
I = I/O chain with permanently closed gate  
W = wait chain with permanently closed gate  
P = partition chain with switchable gate,  
P gates located in Partition Control Block (PCB)  
S = system chain with switchable gate

Figure 26 (Part 2 of 3). Task Status Flags and Resource Gates



Type	Value	Name	Usage
S	8B	USEBND	Gate for LOCK/UNLOCK processing
C	8C	CNCLBND	Gate for subtask to be cancelled
S	8D	SSIDBND	Gate for subsystem id processing
W	8E	RURBND	Gate for LOCK to be freed
S	8F	EXNTBND	Gate for EXTENT processing
P	90	GTVBND	Gate for partition GETVIS
P	91	CDLBND	Gate for CDLOAD
P	92	PFXBND	Gate for PFIX
P	93	IVMTSBND	Gate for IMR data collection
P	94	DYSGVBND	Gate for dyn. space GETVIS
P	95	PTERMBND	Gate for dump handling
<p>Type:    0 = permanently opened gate                  C = permanently closed gate                  I = I/O chain with permanently closed gate                  W = wait chain with permanently closed gate                  P = partition chain with switchable gate,                      P gates located in Partition Control Block (PCB)                  S = system chain with switchable gate</p>			

Figure 26 (Part 3 of 3). Task Status Flags and Resource Gates

## Gating Methods

The different gating methods are described in the following, also the range of application of the different kinds of gates and the function of the POST/RPOST/UNPOST routines in connection with the gate types.

POST/RPOST/UNPOST routines are designed to be called in AMODE 31.

### Setting a Task Ready-to-Run

Tasks selection bit in TSS is turned on.

When a task of a dynamic partition is to be posted, the partition selection bit of the dynamic class (CPCPSS in CPCB) is turned on.

Partitions selection bit in PSS is turned on.

Tasks status flag (TIBRQID) is setup to point to a permanently opened gate (either READY or CONDRDY).

### Setting a Task Not Ready-to-Run

Tasks selection bit in TSS is turned off.

TSS is tested, when it is a zero string:

- When a task of a dynamic partition is to be unposted, the partition selection bit of the dynamic class (CPCPSS in CPCB) is turned off.
- Partitions selection bit in PSS is turned off too.

Tasks status flag (TIBRQID) is setup to point to a closed gate.

### UNPOST Routine

**Note:** The UNPOST routine is always called by a task setting itself to wait.

The parameter to the UNPOST routine is a pointer to the corresponding resource descriptor. In some cases an ECB (or any other) address is in the caller's register R1 which will be passed from the UNPOST to the RPOST routine. For this purpose the last three bytes of R1 are stored to the three bytes at label TIBSTATE+1 (located in the TIB).

### RPOST Routine

The RPOST routine is called in order to post one or more tasks enqueued on a resource. Parameter to the RPOST routine is a pointer to the corresponding resource descriptor. In some cases an ECB (or any other) address is in the caller's register R1 which will be used to identify a wait condition: the last three bytes of R1 are compared with the content of the three bytes at TIBSTATE+1.

### POST Routine

POST routine is called to post a special task, which must be waiting for a permanently closed resource with no central wait queue support. It provides a fast post service for example, for I/O bound tasks. The parameter is a TIB pointer instead of a pointer to a resource descriptor. Note that calls to POST and RPOST are not interchangeable. It is necessary to call the right one in order to get a correct result.

### Processing of Conditionally Ready State (CONDRDY)

In combination with resource types PS (Partition wait queue with Switchable gate) and SS (System wait queue with Switchable gate) tasks are posted one at a time. When there are any other tasks enqueued on the resource the posted one becomes the CONDRDY state, which means that it has been posted in order to take a resource. In order to allow later identification the old resource pointer is

saved to tasks TIB. In some situations the task is not able to take the reserved resource and tries to enter any new wait state. When the UNPOST routine detects a task which is conditionally ready and the corresponding resource is not occupied yet it sets up an implicit call to RPOST using the saved resource pointer. Such a way the next waiter from the reserved queue is posted, allowing current task to enter the new wait state.

## Description of Routines

### 1. Using a Permanently Closed Gate with no Wait Queue Implemented (Type P).

This method is used when the waiting routines are known to the posting routine and can, therefore, be posted directly.

*UNPOST routine:*

When the task has a reserved resource RPOST is called. After this tasks status byte is set up to point to the given gate and the task is set not ready to run.

*POST routine:*

Tasks status byte is changed to READY (X'83') and the task is set ready to run.

**Note:** A call to RPOST would not be correct, since there is no possibility implemented to find a waiting routine using the resource descriptor.

### 2. Processing of a Partition Wait Queue with Switchable Gate (Type PS).

This mechanism is used in combination with the partition internal gates (located in the PCBs). It is assumed that the waiting and the posting tasks belong to the same partition.

*UNPOST routine:*

When the task has a reserved resource RPOST is called. After this the gate is closed (if not closed already) and tasks status byte is setup to point to the closed gate. Tasks TIB is inserted to the front of the wait queue. The task is set not ready to run.

*RPOST routine:*

The gate is opened by the posting routine. The queue is scanned and the oldest waiter (when any) is dequeued. Status byte of the task is set to CONDRDY (respectively READY when it was the only task enqueued on the resource). The dequeued task is set ready to run.

### 3. Using a Common Wait Queue and a Permanently Closed Gate (Type CP).

This mechanism is an extension to 1. A wait queue is maintained which queues the TIBs of the waiting routines together. In addition the contents of the waiting routine's and the posting routine's register 1 is used for wait identification.

*UNPOST routine:*

When the task has a reserved resource RPOST is called. After this tasks status byte is setup to point to the given gate. The waiting routine's register 1 is stored to the TIBSTATE field. The task's TIB is inserted at the beginning of the corresponding wait queue. (The header of the wait queue can be addressed via the resource descriptor entry.) The task is set not ready to run.

*RPOST routine:*

A scan of the wait queue is performed. All tasks whose TIBSTATE match the passed contents of the posting routine's register 1 are removed from the queue.

Status bytes of the tasks are changed to READY. The tasks are set ready to run.

4. Using a System Wait Queue and a Switchable Gate (Type SS).

This is an extension to 2. By maintaining a common wait queue, tasks of multiple partitions can be handled.

*UNPOST routine:*

When the task has a reserved resource RPOST is called. After this the gate is closed (if not closed already) and the task's status byte is set up to point to the given gate. The task's TIB is inserted at the beginning of the corresponding wait queue. The task is set not ready to run.

*RPOST routine:*

The gate is opened by the posting routine. The queue is scanned and the partition priorities of all tasks compared. The oldest waiter (when any) from the highest priority partition is dequeued. Status byte of the task is set to CONDRDY (respectively READY when it was the only task enqueued on the resource). The dequeued task is set ready to run.

5. Gating Via a Permanently Closed Gate With the Additional Possibility to Scan for Waiting Routine (Type FP).

This is an extension to 1. It allows fast direct posting as well as a scan for tasks waiting for a specific ECB. It is implemented for two resources: RBWAIT and RBENQ.

*UNPOST routine:*

When the task has a reserved resource RPOST is called. After this task's status byte is set up to point to the given gate. The last three bytes of the caller's register 1 (ECB pointer) are saved into the TIBSTATE field. The task is set not ready to run.

*POST routine:*

Direct posting is supported in order to allow fast posting for example, from I/O bound state. The task's status byte is set to 'ready' with no regard to the contents of TIBSTATE. The dequeued task is set ready to run.

*RPOST routine:*

The task identifier string of the presently active partition (label TIDSTR, located in the PCB) is scanned for a task with the requested status flag. Each task with the given status flag is posted if

- the contents of the posting routine's register 1 is zero or
- the contents of the waiting routine's TIBSTATE is zero or
- the posting routine's register 1 is equal to the contents of the waiting routine's TIBSTATE field.

---

## VSE/AF Dispatcher - Time Slicing (Partition Balancing)

The priority of the partition/dynamic classes can be changed by the z/VSE Job Control or AR **PRTY** command. The command is extended to specify dynamic classes also, for example,

```
PRTY F9,F6,N,P,S=F2,F3,F1
```

where N, P, S are dynamic classes and S is balanced with F2. It is not necessary anymore to specify all partitions/dynamic classes in the PRTY command. Partitions/dynamic classes not specified receive a system defined lower priority. The PRTY command allows to specify one balanced group (via separator '=').

The partition balancing routine as known in VSE/SP is extended to support balanced partitions within dynamic classes in addition to the balanced group that may be given by the PRTY command.

When a dynamic class contains more than one allocated dynamic partition, the partitions within the dynamic class are balanced (time sliced). The time slice value can be modified via the **MSECS** command.

The following paragraphs describe the balancing algorithm.

### Definition

**balanced group** A balanced group is a number of partitions and/or dynamic classes with a given time slice (entered by the MSECS command). After the time slice is exhausted a partition may be moved to the lowest priority in that group (described later). The members of the balanced group (only one group in the system) are determined by

1. the PRTY command, for example,

```
PRTY ...,BG,F1=S=F3,F4,F5,...
```

In the example F1, S and F3 are members of the balanced group.

CPUTX	CPU time used by partition/dynamic class x
HTIME	high time value for job accounting (8 hours)
MSECS	user-specified limit for changing priorities (entered by the MSECS command)
MAXMSECS	10*MSECS, max. possible CPU timer value
PBALTIME	partition balancing time
RUNTIME	partition time counter (reset during update of job accounting counters)
SUMCPUT	CPU time used by a balanced group
TSLICE	time set into CPU timer

## GETPRTY / SETPRTY (SVC 57)

The PRTY command process calls the GETPRTY and SETPRTY services. The priority processing routines (GETPRTY and SETPRTY) are extended to support dynamic classes, too.

The external interface of the SETPRTY routine is not changed. Only the internal processing is adapted.

The GETPRTY macro is extended to get the total priority string (static partitions and dynamic classes) in addition to the string of static partition priorities.

### **SETPRTY - Internal Processing**

#### *1. No balanced Group Specified*

The TSCP queues are rearranged dependent on the given priority list (PRTYLIST). Balancing indications of a former PRTY command are reset and a basic time slice value is set, when no more balanced groups are active.

#### *2. One Balanced Group Specified*

a. The following is done for all members of the group:

- the partition/dynamic class is marked as balanced (BALANCED set in PCBFLAG)
- accounting on partition/dynamic class base, not on system base (partition PCB pointer moved to PCBJAPTR) is indicated
- reset partition/dynamic class time counter (RUNTIME in PCB),
- reset partition/dynamic class balancing time (PBALTIME in PCB),

b. rearrange selection strings,

c. save lowest balanced partition/dynamic class (ALBALPCB)

d. when the priority of a partition/dynamic class has changed, set PCEPRTYC in PCEFLAG. This interface is used for communication with the VSE/POWER partition. VSE/POWER resets the flag.

e. when no balanced group was active up to now,

- set CPU timer with time slice specified via MSECS command
- set time slicing active (PBALACT in SUPVFLAG)

**Note:** The first part of the CPCB (dynamic class PCB) is an overlay of the PCB, therefore only PCB fields are shown to describe the balancing algorithm.

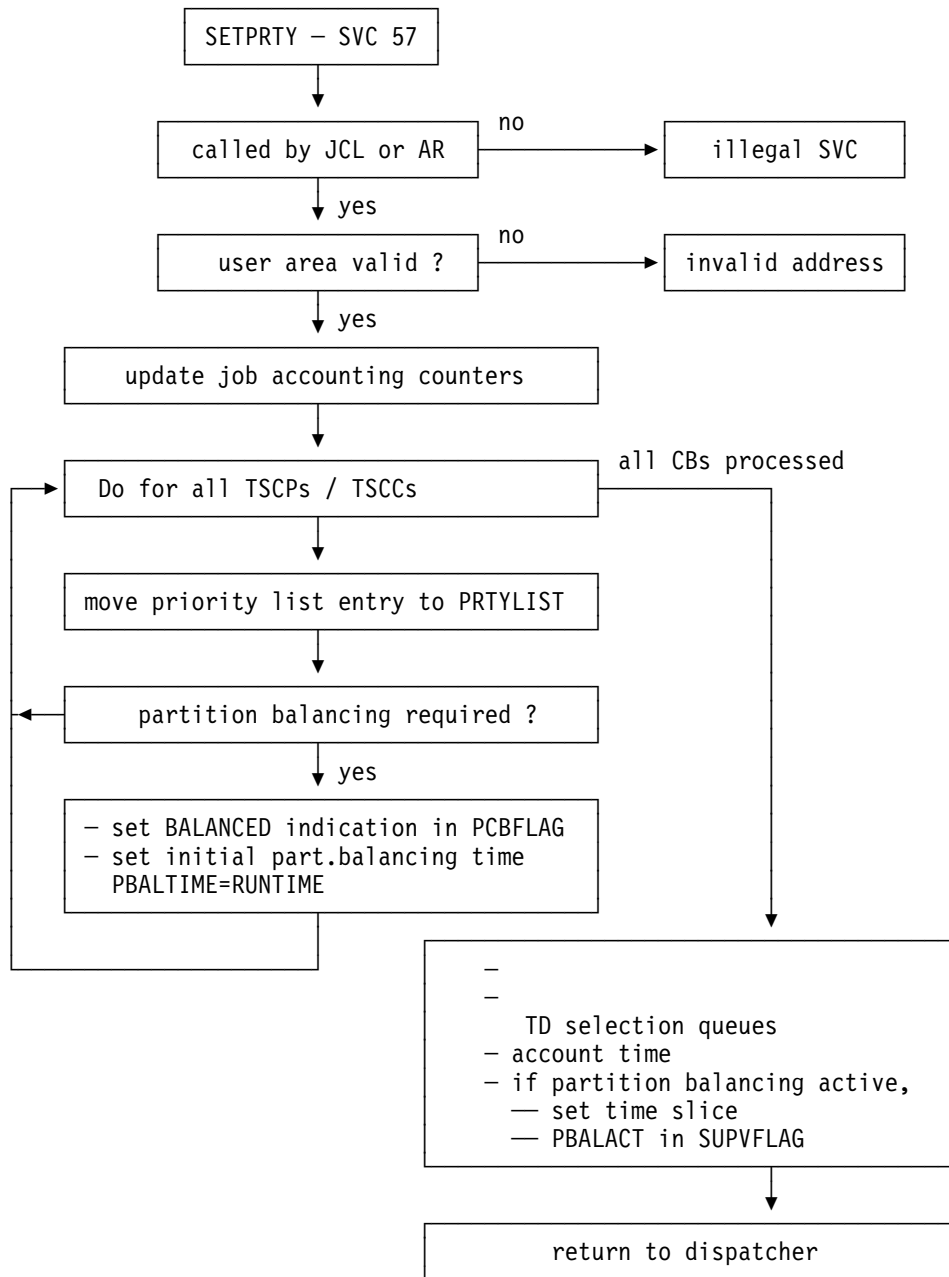


Figure 27. SETPRTY Overview

### GETPRTY

See also the GETPRTY macro description.

### Partition Balancing

The dispatcher (DSP) system task sets up clock comparator time intervals to inspect the partitions of the balancing group. The time interval can be modified via the AR and JCL MSECs command.

The partition balancing routine inspects the CPU time for each partition of the balancing group and decides how to rearrange the priority of the partitions.

If partition balancing is specified for static and dynamic classes (via equal signs in the PRTY command), static and dynamic partitions will receive the same time slice.



In the past the dynamic class (with all its partitions) got the same time slice as a static partition. When the priorities are to be rearranged, the whole TSCP ready-to-run queue will be locked, that is no other CPU can select elements from the TSCP ready-to-run queue.

### **Interrupt Processing and Dispatching**

The accounting interval is marked via the ARUNTIME pointer (PCB pointer of the balanced partition), that is accounting will be done only, if accounting interval is switched. That may occur before dispatching of a task or for example, after an external, I/O interrupt, etc. If time accounting has to be done, the used CPU time of a balanced partition is determined and added to the partition time counter (RUNTIME in PCB). For dynamic partitions the dynamic class CPU time is also updated (CPCBRUNT in CPCB).

### **Time Slice Exhausted (Priority Change Processing)**

When the time slice is exhausted, the CPU timer will present an external interrupt. The external interrupt handler calls the CPU timer interrupt routine (EXTCPUT).

- Turbo Dispatcher

When the time slice is exhausted, the clock comparator will present an external interrupt. The external interrupt handler posts the dispatcher (DSP) system task.

#### ***Partition Balancing Not Active***

When partition balancing is not active (PBALACT not set), the DSP system task checks if any service is requested, sets up the basic time slice (MAXMSECS) and waits for expiration of the clock comparator.

#### ***Partition Balancing Active***

When partition balancing is active, the CHNGPRTY routine (for the Turbo Dispatcher located in the TD module \$IJBDSPT) continues as follows:

1. When total time used for balancing (TOTTIME) is exhausted,  
TOTTIME >= HTIME  
reset TOTTIME and indicate job accounting update necessary,
2. First process the balanced group specified by PRTY command:  
When it is necessary to rearrange the priorities,
  - a. update TSCP queues,
  - b. set new basic balancing value (PBALTIME),
  - c. update priority figures.
3. Do for all dynamic classes, where balancing is active:  
When it is necessary to rearrange the priorities,
  - a. update TSCP queues,
  - b. set new basic balancing value (PBALTIME),
  - c. update priority figures.
4. Account used time and set new time slice,
5. Call UPDJA routine, if job accounting update is necessary.

#### ***How to Determine the Partition/Dynamic Class to Be Rearranged***

1. Look for the highest priority partition/dynamic class within group (=partition/dynamic class to be removed), such that

CPUTX >= SUMCPUT  
where CPUX=used time of partition/dynamic class=RUNTIME-PBALTIME

2. Do not rearrange partition priorities, when

MSECS > SUMCPUT

3. Rearrange partition priorities, when

MSECS <= SUMCPUT

### How to Determine a New Time Slice

$$\begin{aligned} \text{TSLICE}(\text{new}) = & \\ & \text{minimum}(\text{MAXMSECS}, \text{TSLICE}(\text{new}), \text{MSECS} + \text{TSLICE}(\text{old}) - \text{SUMCPU}) \\ & \text{where } \text{MAXMSECS} = 10 * \text{MSECS} \end{aligned}$$

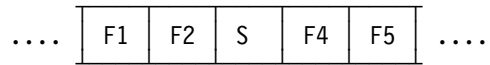
### Example

members of balanced group: F1, F2, S, F4, F5

Priority command: PRTY ..., F1=F2=S=F4=F5, ...

CPU intensive dynamic class to be removed: S

priority before move: (low → high)



priority before move: (low → high)



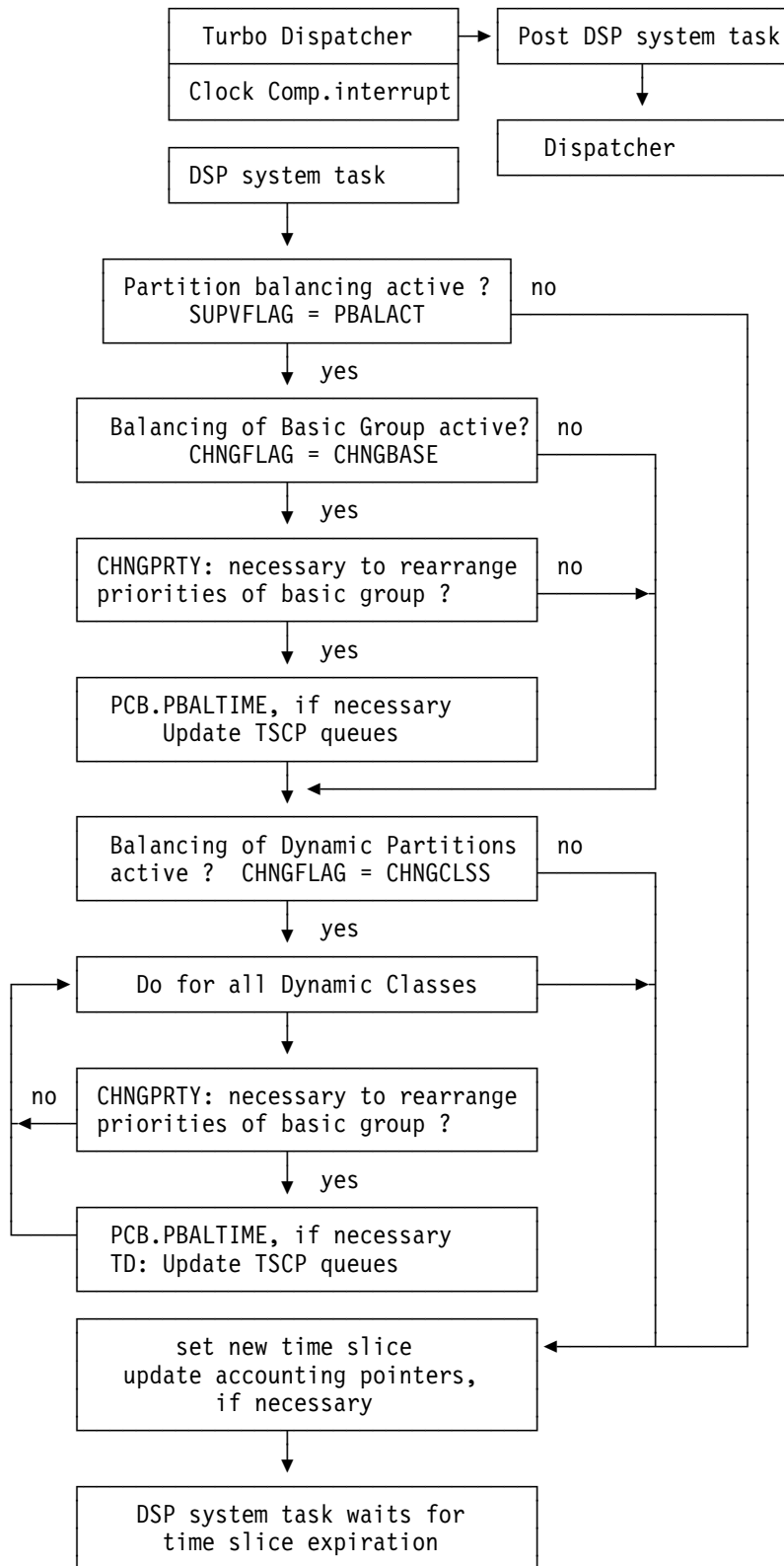


Figure 28. Partition Balancing Routine Overview

## z/VSE Dispatcher - Task Termination

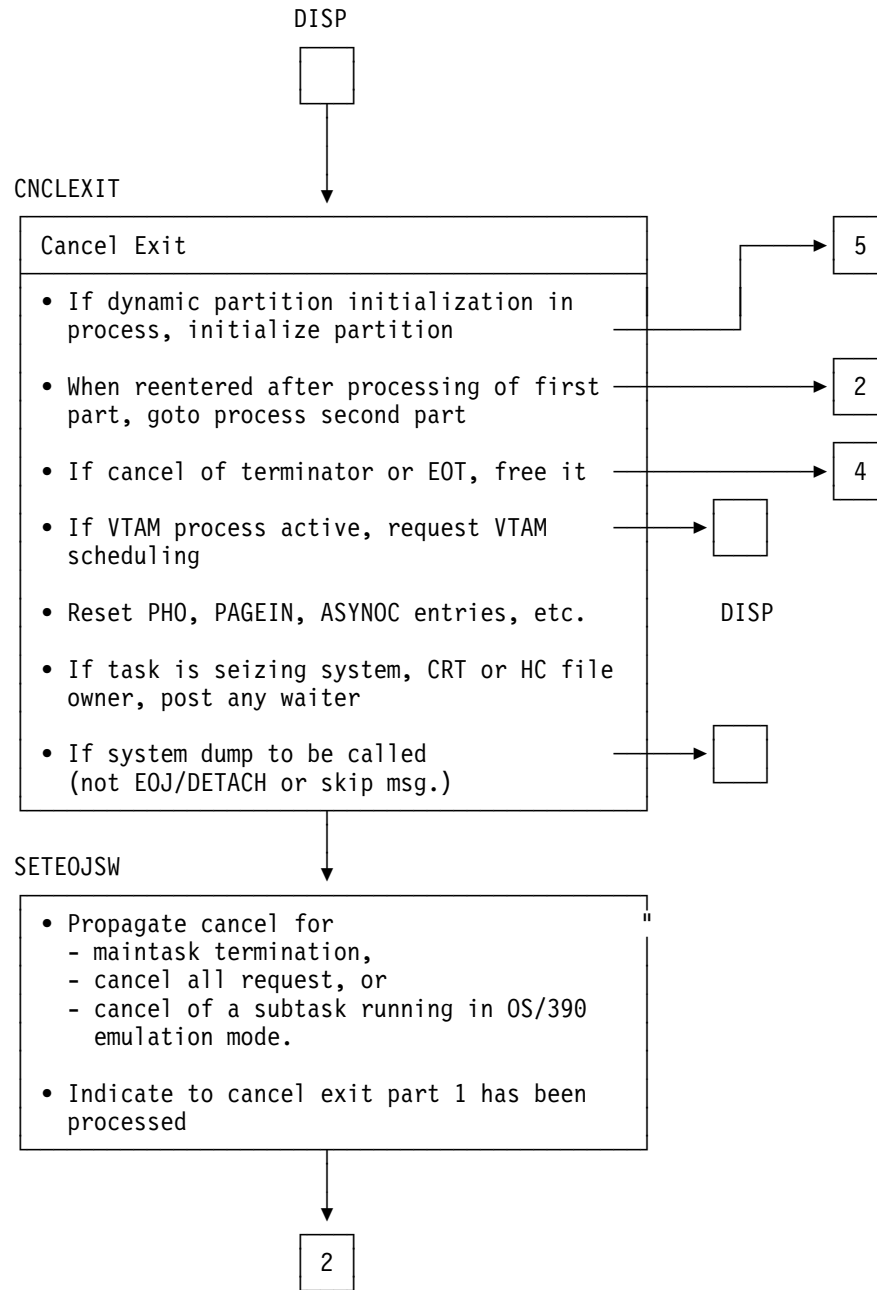


Figure 29 (Part 1 of 3). Supervisor General Exit, Cancel Exit

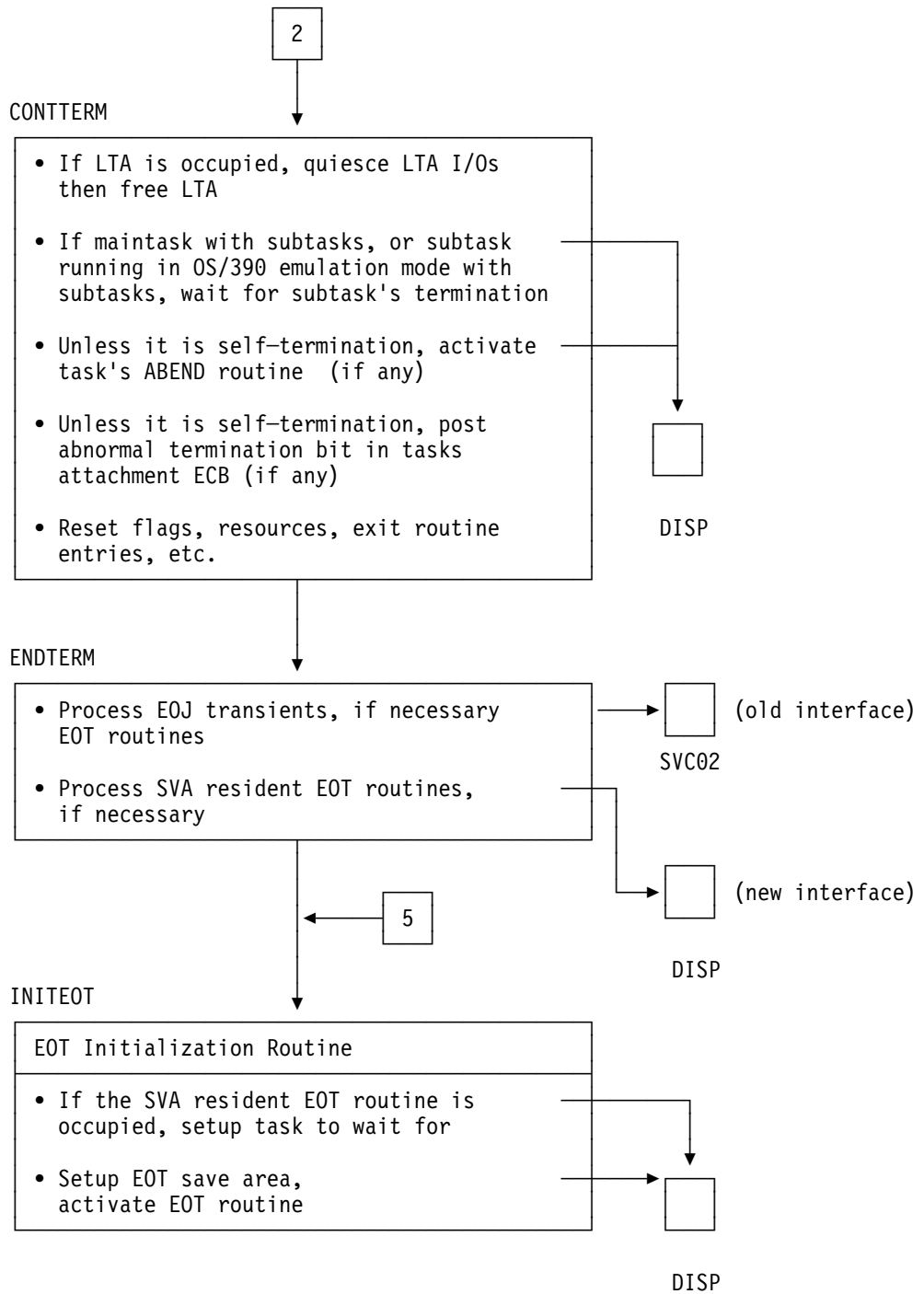


Figure 29 (Part 2 of 3). Supervisor General Exit, Cancel Exit

EOTRTRN

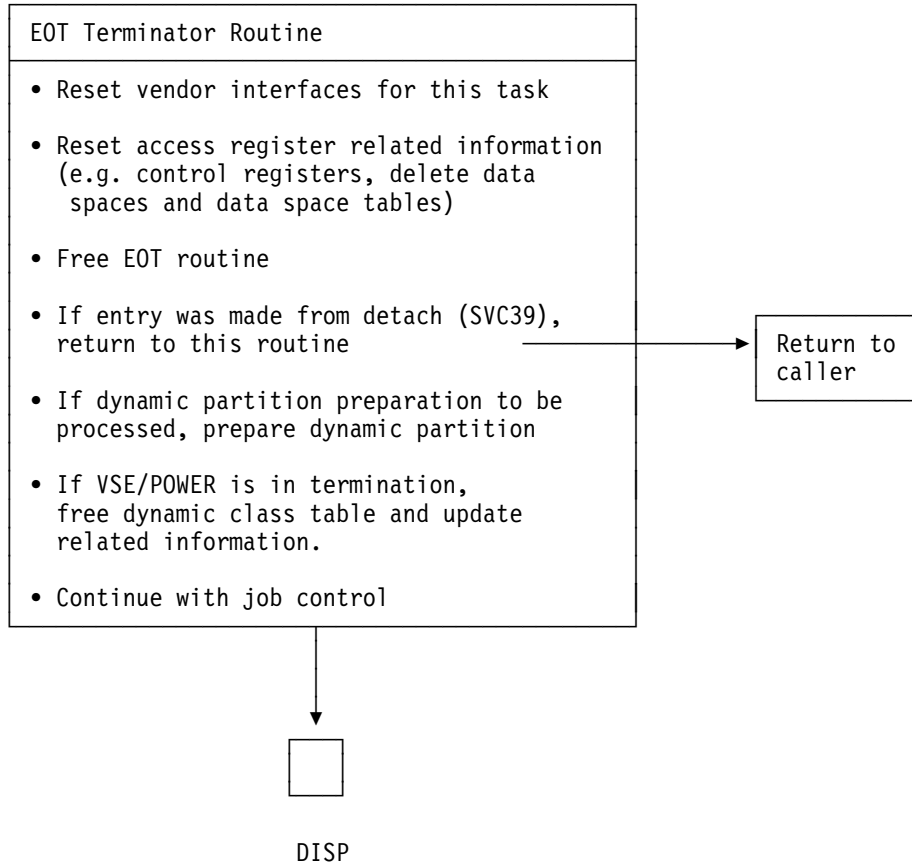


Figure 29 (Part 3 of 3). Supervisor General Exit, Cancel Exit

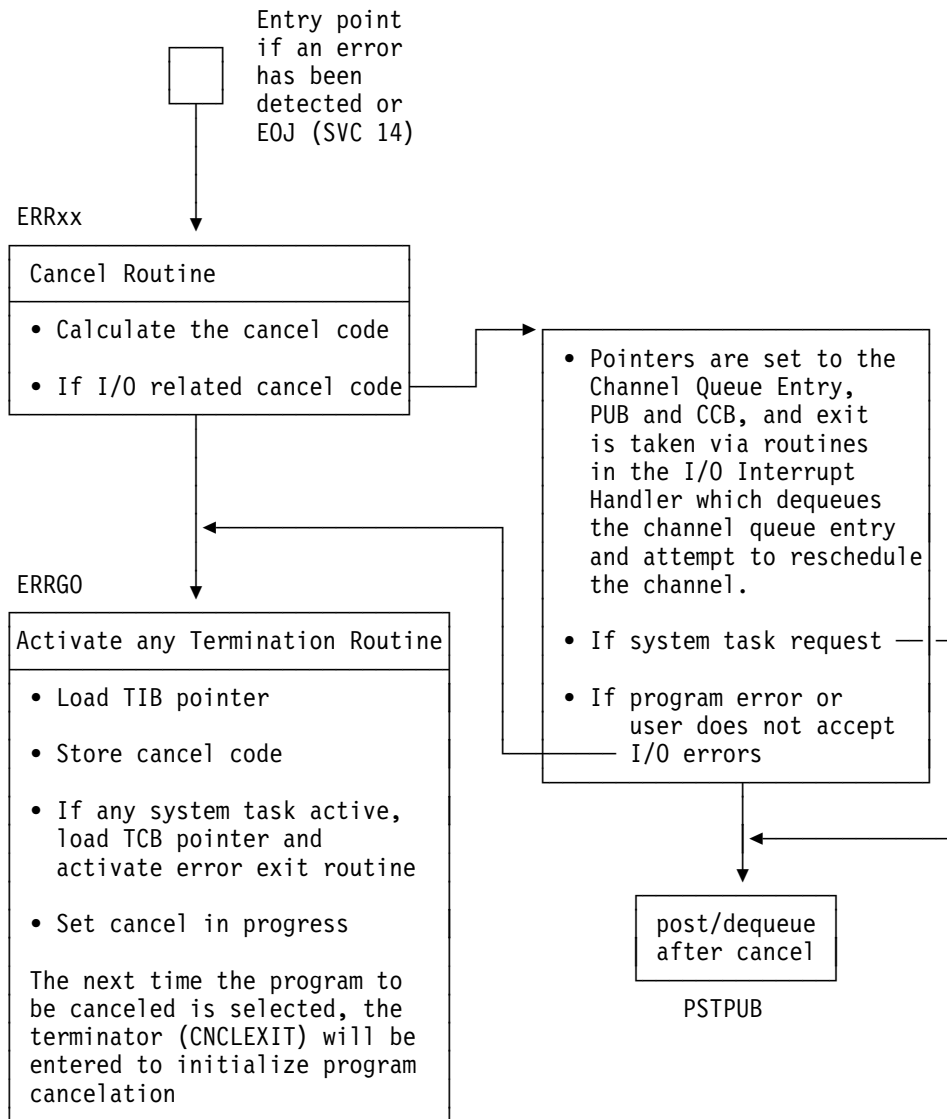


Figure 30. Cancel Routine



---

## z/VSE Dispatcher - System Dump Interfaces

### Overview

The System Dump routine (phase \$IJBSDMP) consists of two parts

1. the message writer for abnormal termination messages and
2. the storage dump routine.

The System Dump routine will be called

- to process a dump service; the dump services use either SVC 2 (DUMP, JDUMP, PDUMP macros) or SVC 123 (SDUMP, SDUMPX macros) to request a call of the System Dump routine,
- during abnormal task termination.

This paragraph describes the supervisor interfaces implemented for System Dump routine initialization and termination.

Since VSE/ESA 1.3.0 the System Dump routine can execute in parallel. Only tasks within the same partition will be gated, that is tasks have to wait, if another task of the same partition occupies the System Dump routine. Only in very rare cases, when no GETVIS space is available the System Dump routine is gated system wide.

The following interfaces are used:

- a resource descriptor (gate) in Partition Control Block (PCB)
- a supervisor / System Dump routine communication area (SAACOMM) per partition, which will be allocated during the partition allocation process, and deallocated during partition deallocation. The mapping macro MAPSAACM describes the layout of SAACOMM.
- a pointer in PCB (PCBSAAPT), that holds the address of the SAACOMM.
- one master SAACOMM as part of \$IJBSDMP, that is used when the GETVIS space is exhausted
- the System Dump routine will return to the supervisor via SVC 14 (and no longer via SVC 11)

The System Dump routine (\$IJBSDMP) will be loaded during IPL. The supervisor gets the address of \$IJBSDMP from the supervisor subdirectory (SVASVDL).

**Note:** The Partition Debug Facility will also use the system dump initialization and termination routines.

## Flow of Control (Normal and Abnormal Termination)

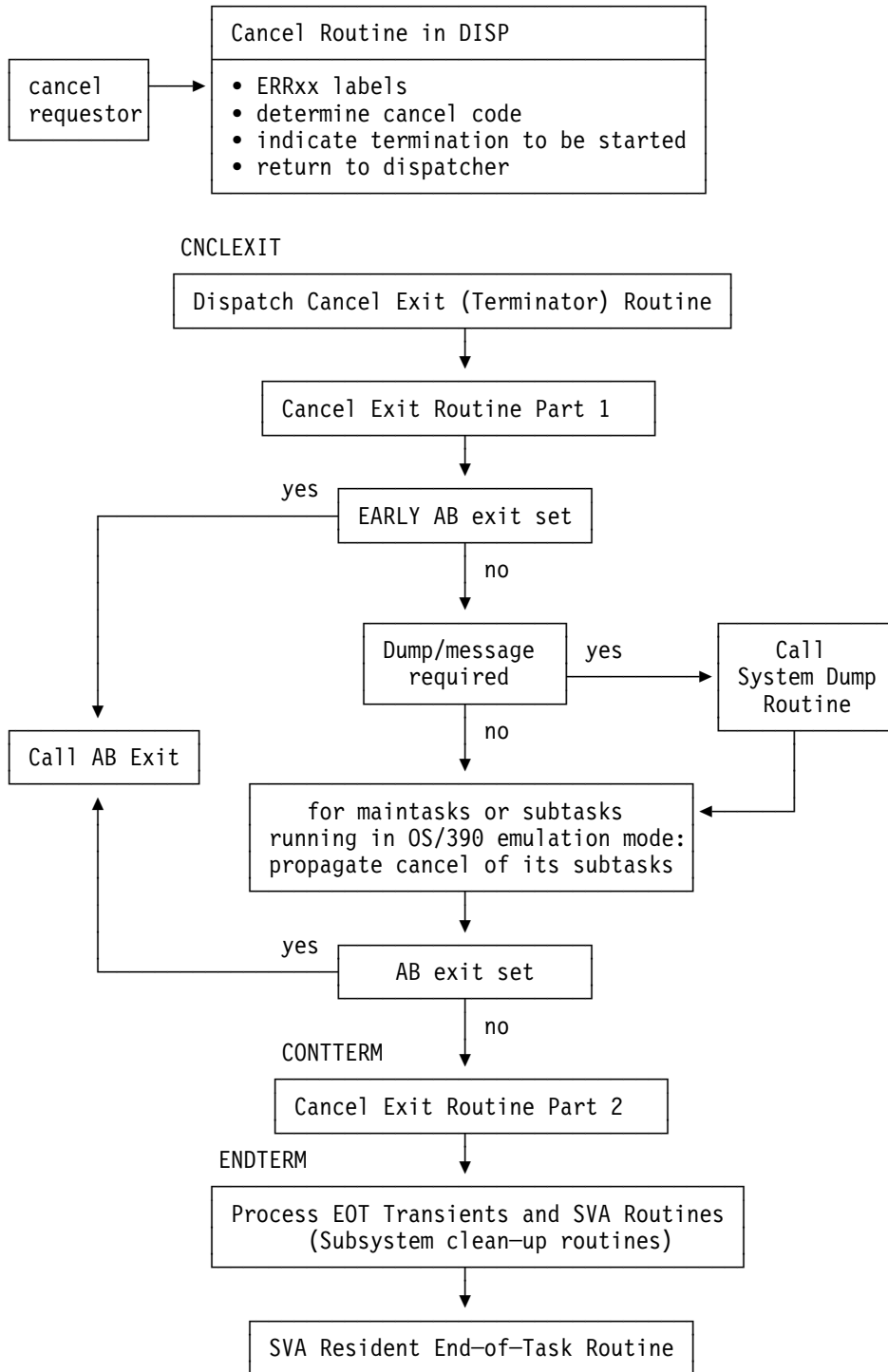


Figure 31. Flow of Control for Normal and Abnormal Termination

## Flow of Control (System Dump Routine Initialization / Termination)

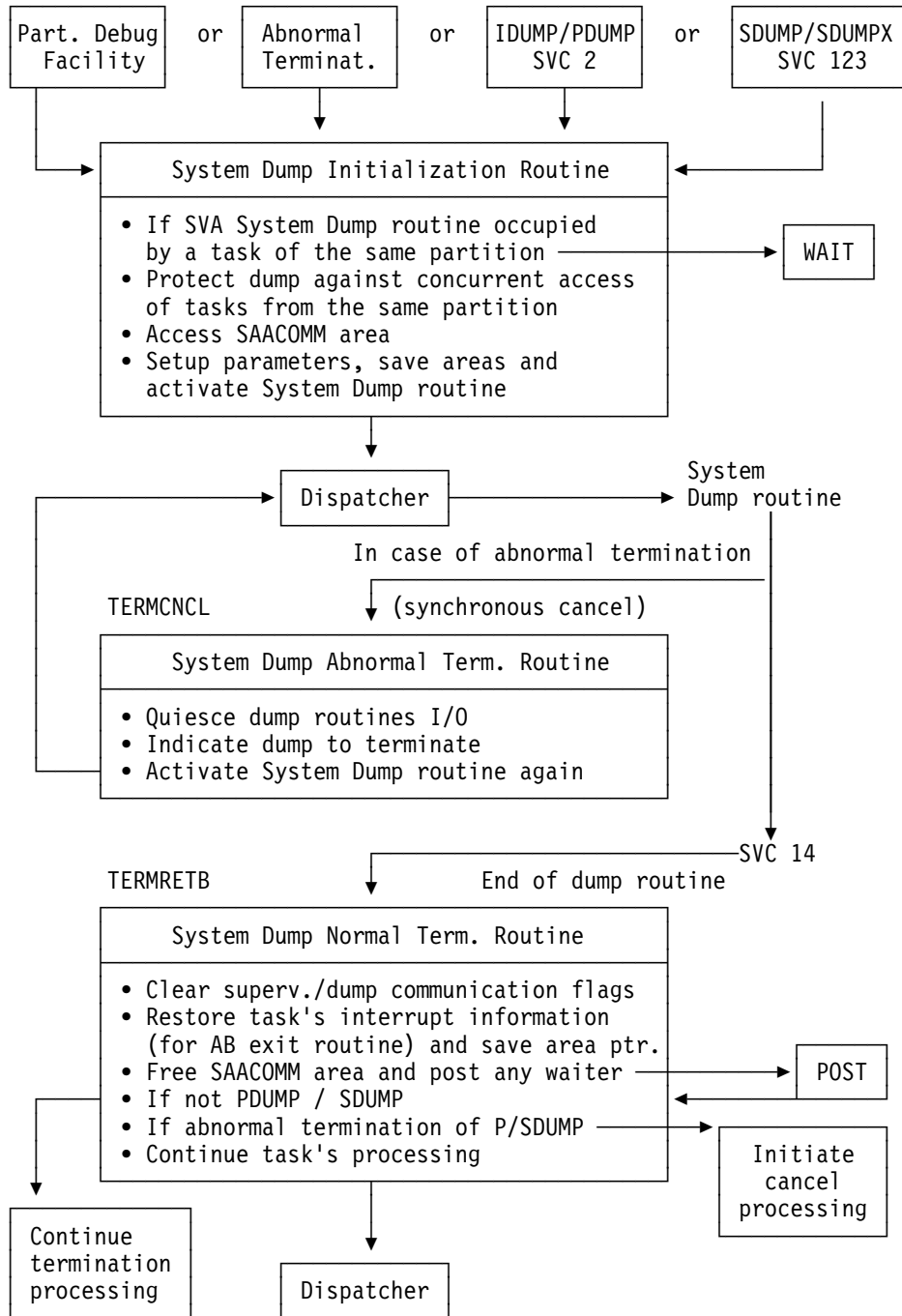


Figure 32. Dump Routine Initialization and Termination

## System Dump Phase Structure

A System Dump SAACOMM area is located at offset 0 of \$IJBSDMP. This communication area is called master SAACOMM, indicated in SAAMCB of SAAFLAG2. The address of the first instruction of \$IJBSDMP is located at SAADSTRT. The master SAACOMM will be used,

- if a GETVIS request fails.  
Then the System Dump routine is gated system wide (SRQTERM).

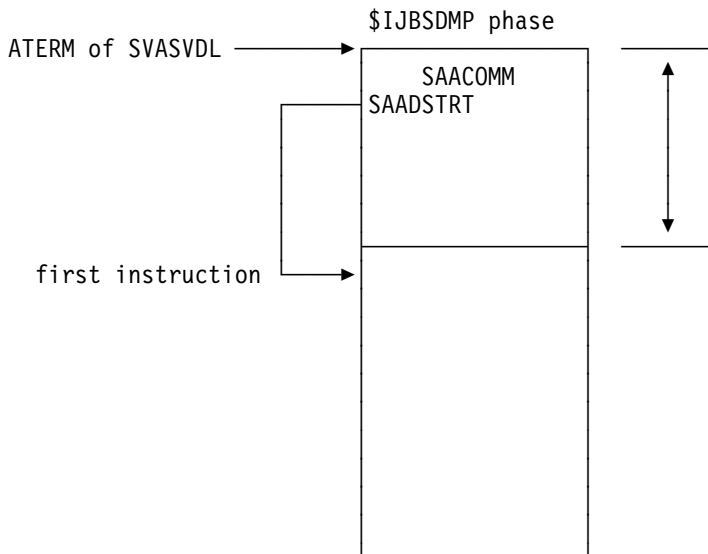


Figure 33. System Dump Phase Structure

## Activation of the System Dump Routine

The System Dump initialization routine will be activated

1. during abnormal termination, if message output (for example, cancel messages) or dump output is wanted.
2. by a DUMP, JDUMP or PDUMP request via SVC 2. This will be indicated in the flag byte SAAFLAG,
3. by a SDUMP or SDUMPX request via SVC 123. This will be indicated in the flag byte SAAFLAG.
4. by a the Partition Debug Facility (e.g. for initialization of the facility or when active after a PER interrupt). This will be indicated in the flag byte SAATFLAG. The Partition Debug Facility will be processed like a PDUMP request, the process will not be contained in the following paragraph.

### System Dump Initialization Routine

This paragraph describes the steps necessary before activation of the System Dump Routine.

1. identify the initialization routine as reentrant routine (RID = REENTRID)
2. protect the System Dump routine against concurrent access of tasks belonging to the same partition, that is the System Dump routine has to be gated on parti-

tion base. Therefore it was necessary to introduce a new partition gate (PCBRBTRM) within resource SRQPTERM.

- If this gate is free,
  - the task has to set the gate to "notfree" and can proceed.
- If the gate is not free,
  - the task has to wait until it is posted by "System Dump Normal Termination Routine" on page 83. For PDUMP, SDUMP and SDUMPX requests RESVCX has to be called, for abend dumps UNPOST.

**Access the Supervisor/System Dump Communication Area (SAACOMM):** The pointer to this area (PCBSAAPT) can be obtained from the PCB. The SAACOMM will be allocated in pageable system GETVIS area (24 bit) during partition allocation.

### **SAACOMM Update**

1. Indicate the task as terminator owner (TERMACT in TIBFLAG1) and make the SAACOMM addressable.

The SAACOMM area is pageable, therefore page faults may occur.

2. *Update SAACOMM*

The fields are not cleared. Which fields are valid in SAACOMM, can be obtained by the flag bytes SAAFLAG and SAAFLAG1.

The following fields may be updated:

- Set in use indication (SAAIUSE in SAAFLAG1) and initialize SAANAME.
- If LTA active for this task (LTAECT+LTAOWNER in TIBFLAG1),
  - store the LTA save area pointer (TCBSAVE) to SAALTAPT,
  - store the problem program save area (PIBSAV2 of ARPIB) to SAAPPSPT,
  - set SAALTA and SAAPPA in SAAFLAG,
- If not (LTAECT+LTAOWNER in TIBFLAG1),
  - move TCBSAVE (actual problem program save area) to SAAPPSPT,
  - set SAAPPA in SAAFLAG,
- Move first cancel code (TIBCNCL) to SAACCL1,
- Move second cancel code (TIBCNCL2) to SAACCL2,
- Move interrupt information (INTINFO of TCB) to SAAINTC,
- If abend dump, move additional message information to SAADMSGI (if available) and set corresponding flag (SAAMSGI in SAAFLAG1)
- If PDUMP, SDUMP or SDUMPX,
  - set SAAPDUMP for PDUMP in SAAFLAG,
  - set SAASDUMP for SDUMP and SDUMPX in SAAFLAG,
- Move access registers to SAAARSAV and set corresponding flag SAAARA in SAAFLAG1, if available.

**Note:** SAAARSAV will be restored after return from System Dump routine.

3. TCBSAVE:=pointer to SAARSAV, make the System Dump save area (at SAARSAV) to current save area.

4. If abnormal termination is caused by supervisor,

- move supervisor special save area to SAASSAVE, and
- set SAAPCA in SAAFLAG,

5. Set System Dump cancel ECB (part of TIBFLAG4) address to SAACNECB. The System Dump may wait for this ECB during dump process and will be posted, if for example, the operator cancels the partition.
6. continue with "Make System Dump Dispatchable."

#### ***Make System Dump Dispatchable***

1. Move a new PSW to the System Dump save area. The new PSW contains the start address of the System Dump routine (SAADSTRT).
2. Move SAACOMM pointer to dump routines register 0.
3. Move TERMSERV pointer to dump routines register 1.
4. If message to be printed only (PRTMSG in TCABFL1),
  - set SAANODMP in SAAFLAG1
5. If PERACT in PCBFLAG,
  - Enable program event recording,
6. Dispatch System Dump

## **Return from (Termination of) System Dump Routine**

This chapter describes the return from / termination of the System Dump Routine. Three cases may occur:

1. Return from System Dump routine because of GETVIS error (via SVC 14 - EOJ).
2. Abnormal termination (System Dump routine cancelled).
3. Normal termination (via SVC 14 - EOJ).

### **Return from System Dump Routine (GETVIS Error)**

When the System Dump routine detects a GETVIS error,

1. it sets flag SAANGETV in SAAFLAG1 to indicate GETVIS error,
2. returns to the supervisor via SVC 14.

Now the supervisor processes the following steps:

1. Use system gating and master SAACOMM
2. Partition's SAACOMM address is saved in field TERMSCSA,
3. Set SAASYS in partition's SAACOMM,
4. PCBSAAPT and PCBMSAAE is set,
5. Set task's dispatching priority to highest user task priority,
6. Move partition's SAACOMM to master SAACOMM,
7. Continue with "Make System Dump Dispatchable."

## System Dump Abnormal Termination Routine

The System Dump abnormal termination routine is called, whenever the System Dump cancels, the processing continues with the following steps:

1. Quiesce LTA's I/O (SVC03LTA) of System Dump routine,
2. Set RID to REENTRID (allow page faults)
3. If multiple cancel (SAACNCL already set),
  - indicate 'TERMINATOR ROUTINE CANCELLED' message has to be written by end-of-task routine,
  - continue with "System Dump Normal Termination Routine,"
4. Move cancel code (TIBCNCL2 for abend dump, TIBCNCL for PDUMP / SDUMP/ SDUMPX) into SAACNCL3,
5. Set SAACNCL flag in SAAFLAG,
6. Move current status (PSW and registers to SAASSAVE) and set SAAPCA flag in SAAFLAG,
7. If master SAACOMM used,
  - set task's dispatching priority to highest user task priority,
8. Continue with "Make System Dump Dispatchable" on page 82.

## System Dump Normal Termination Routine

This paragraph describes the steps necessary after normal termination of the System Dump routine via SVC 14.

**Note:** The former return from System Dump routine via SVC 11 is replaced by SVC 14 (EOJ macro).

1. Set RID to REENTRID (allow page faults)
2. Restore current save area pointer
  - If SAALTA in SAAFLAG,
    - move SAALTAPT to TCBSAVE,
  - Otherwise move SAAPPSPT to TCBSAVE,
3. If master SAACOMM allocated to partition (PCBMSAAE in PCBSAAPT)
  - post any system gate waiters
  - restore partitions SAACOMM address (TERMSCSA -> PCBSAAPT)
  - clear TERMSCSA
  - reset SAAFLAG and SAAFLAG1 flags of current SAACOMM,
  - reset high priority indication
4. Post any partition gate waiters
5. Reset dump routine active (TERMACT in TIBFLAG1)
6. Restore interruption code (for AB exit routines) SAAINTC to INTINFO of TCB,
7. Restore access registers from SAAARSAV,
8. Check for VTAM application processing
9. If system abend request (SAAABEND in SAAFLAG1),
  - reset partition's SAAFLAG and SAAFLAG1 flags
  - continue with task's termination
10. For PDUMP/SDUMP/SDUMPX processing continues as follows:

- Reset partition's SAAFLAG and SAAFLAG1 flags
- If PDUMP/SDUMP/SDUMPX cancelled (TIBCNCL not zero),
  - set TIBDMPCN flag in TIBFLAG4 to avoid call of System Dump routine  
TIBDMPCN will be reset during the termination process,
  - initiate cancel processing
    - set FETCHEOJ in TIBFLAG,
- Check for delayed IT and OC (maintasks only) interrupts
- Return to dispatcher

## Synchronous and Asynchronous Cancel

In the past no difference was made between synchronous and asynchronous cancel.

First the difference between these two cancel situations:

- |                     |   |
|---------------------|---|
| <b>synchronous</b>  | all cancel conditions caused by the dump routine itself, for example, program check.      |
| <b>asynchronous</b> | all cancel conditions caused by another task or partition, for example, AR CANCEL logid . |

### Synchronous Cancel Conditions

All cancel conditions caused by the dump routine itself are so called synchronous events. For more information see “System Dump Abnormal Termination Routine” on page 83.

**Note:** In case of synchronous cancel the communication - supervisor to dump - is done via the current SAACOMM area.

### Asynchronous Cancel Conditions

All cancel conditions caused by other tasks or partitions are so called asynchronous events. To avoid inconsistent states of the System Dump routine the cancel process has to be delayed until the System Dump routine detects the request. The cancel propagation routine (RDYCNCL) in the dispatcher will delay cancel requests whenever the System Dump routine (TERMACT in TIBFLAG1) is active. The following actions have to be taken:

1. Request dump to terminate,
  - indicate cancel request pending (TIBDMPCN of TIBFLAG4),
  - move requestors cancel code into TIBCNCL3, if abend dump is taken. This causes the end-of-task routine to write the 'TERMINATOR ROUTINE CANCELED' message, if System Dump is not able to write the corresponding message. If the System Dump routine was called because of a PDUMP, SDUMP or SDUMPX request, TIBCNCL is set.
  - post the System Dump routine, TIBDMPCN is used as ECB traffic bit.
2. The System Dump routine checks from time to time the TIBDMPCN field via the new terminator service TERMSERV (see “System Dump Services (TERMSERV)” on page 85). If a cancel condition is returned, the dump routine terminates its processing (may be with a cancel message) with SVC 14 (normal return to cancel exit).

**Note:** In case of asynchronous cancel the communication - supervisor to dump - is done via the new TERMSERV services.



## System Dump Services (TERMSERV)

Currently two services are available

- check, if cancel of System Dump is requested
- reset task's dispatching priority to its original priority,

The address of the TERMSERV entry point will be passed in register 1 during System Dump activation. When TERMSERV is to be called, a function code has to be set in register 0.

### *Input Registers:*

Register 0                    function code  
Register 15                 TERMSERV routine address = base address

### *Output Registers:*

Register 15                 may contain a return code

### *Work Registers:*

Register 0 and 1

### **Function Code 0: Check if System Dump to be Canceled**

When System Dump cancel is requested, TIBDMPCN in TIBFLAG4 will be set in RDYCNCL routine. The TERMSERV service checks this flag and if set,

- resets TIBDMPCN,
- sets SAACNCL in SAAFLAG1,
- moves the cancel code into SAACNCL3
- clears TIBCNCL3 to avoid the end-of-task message,
- set return code of 4 in register 15.

TERMSERV returns with return code 0 otherwise.

### **Function Code 4: Set System Dump Owner to its Dispatching Priority**

The service schedules the dispatcher exit (TERMSRES). The exit sets the System Dump Owner to its original dispatching priority.

### **Register Conventions for System Dump Activation**

The System Dump initialization routine initializes the System Dump save area (located in SAACOMM at label SAARSAVE), that is

- the PSW of System Dump start address (SAADSTRT)
- register conventions at activation:
  - Register 0**                    address of current SAACOMM
  - Register 1**                    address of terminator service routines
  - Registers 2 - 15**             unpredictable.

## z/VSE Dispatcher - VSE/ICCF Support

Since VSE/ESA 1.3.0 VSE/ICCF runs in VSE/AF subtasks, that is CICS will occupy the VSE/AF maintask, even if VSE/ICCF is active within the CICS partition.

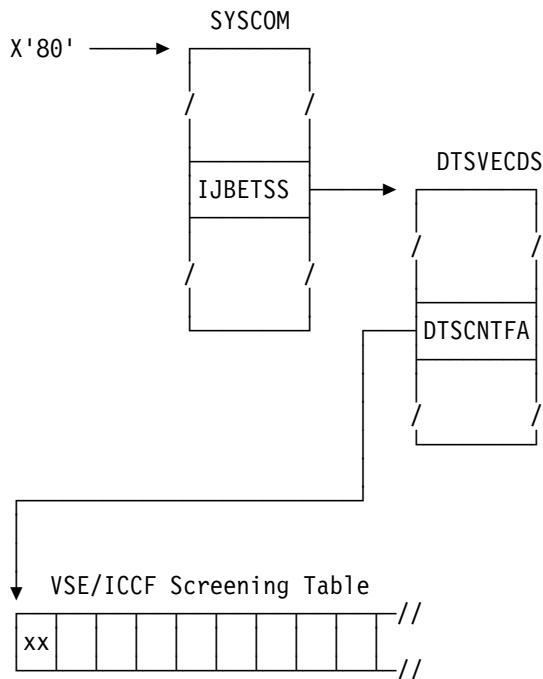
This section describes how the VSE/AF supervisor supports VSE/ICCF running in subtasks.

### VSE/ICCF Pseudo Partition: Dispatching

The ICCF pseudo partitions have to be dispatched with the lowest priority within the CICS partition, where VSE/ICCF is activated. The priority has to be lower than the priority of the CICS maintask and all other subtasks. When VSE/ICCF identifies a VSE/AF subtask as a pseudo partition (via MODFLD service), the VSE/ICCF pseudo partition is removed from the TSCT ready-to-run chain and is enqueued to a lower priority than the maintask (behind the maintask TSCT).

### VSE/ICCF Pseudo Partition: ICCF Screening

VSE/ICCF allocates a 256 byte screening table in PFIxed system GETVIS storage (RMODE 24). A byte of the screening table is called SVC screening byte. Each byte corresponds to a SVC number. VSE/ICCF has to initialize the SVC screening bytes and to establish addressability to the table before ICCF screening is activated. The address of the table has to be stored into a new field of the VSE/ICCF vector table:



`xx` = ICSVCIGN, ICSVC or ICSVCERR  
(see "Contents of VSE/ICCF Screening Byte:" on page 87)

Figure 34. VSE/ICCF Screening Table - Control Block Relationship

<b>IJBETSS</b>	in SYSCOM holds address of the VSE/ICCF vector table (DTSVECDSD)
<b>DTSVECDSD</b>	mapping of VSE/ICCF vector table
<b>DTSCNTFA</b>	address of VSE/ICCF screening table, initialized by ICCF, removed during VSE/ICCF shutdown.

***Contents of VSE/ICCF Screening Byte:***

<b>ICSVCIGN (=0)</b>	VSE/ICCF interrupt handler will not handle the SVC,
<b>ICSVC (=4)</b>	VSE/ICCF interrupt handler wants to handle the SVC,
<b>ICSVCERR (=8)</b>	illegal SVC in the VSE/ICCF environment

The VSE/AF supervisor first level SVC interrupt handler will pass control to the VSE/ICCF first level interrupt only, if the SVC screening byte contains value ICSVC (=4) and the VSE/ICCF task's screening flag (ICCF SVC in TCBFLAGS) is set.

## z/VSE Dispatcher - Partition Preparation and Cleanup

Before a VSE/POWER job is executed, partition preparation will be started within the partition. When VSE/POWER job ends, partition cleanup will be called. The following paragraphs describe the process for static and dynamic partitions.

### Static Partition Preparation and Cleanup

#### Normal Processing

- Preparation (DYNCLASS ID=PREPARE) will execute at the beginning of a VSE/POWER job, Cleanup will execute at the end of each VSE/POWER job for which preparation was done.
- Preparation/cleanup (P/C) will not execute in the VSE/POWER partition instead it will run in the VSE/POWER-controlled partition being prepared/cleaned up. Prepare will be called by Job Control before processing of the first Job Control statement. Job Control calls cleanup. P/C is considered to be part of the user's job.
- VSE/POWER posts Job Control for preparation and cleanup. "Do prepare" and "Do cleanup" will be indicated in the PCE control block. The "Do prepare" indication will be reset by the DYNCLASS ID=PREPARE service. The "Do cleanup" indication must not be reset.
- Job Control sets up a new read request after cleanup.

Action to be taken	Initiated by	Function to be called
1) Job submission to VSE/POWER	II	
2) Request preparation	VSE/POWER	
3) Schedule the job in this partition	VSE/POWER	
4) Initiate preparation	Job Cont.	DYNCLASS ID=PREPARE
5) Process preparation	superv.	
6) VSE/POWER job is in execution		
7) * \$\$ E0J reached		
8) Request partition cleanup	VSE/POWER	
9) Initiate cleanup	Job Cont.	DYNCLASS ID=CLEANUP
10) Process cleanup	superv.	
11) Wait for next VSE/POWER job	Job Cont.	

Figure 35. Scenario: Static Partition Preparation and Cleanup

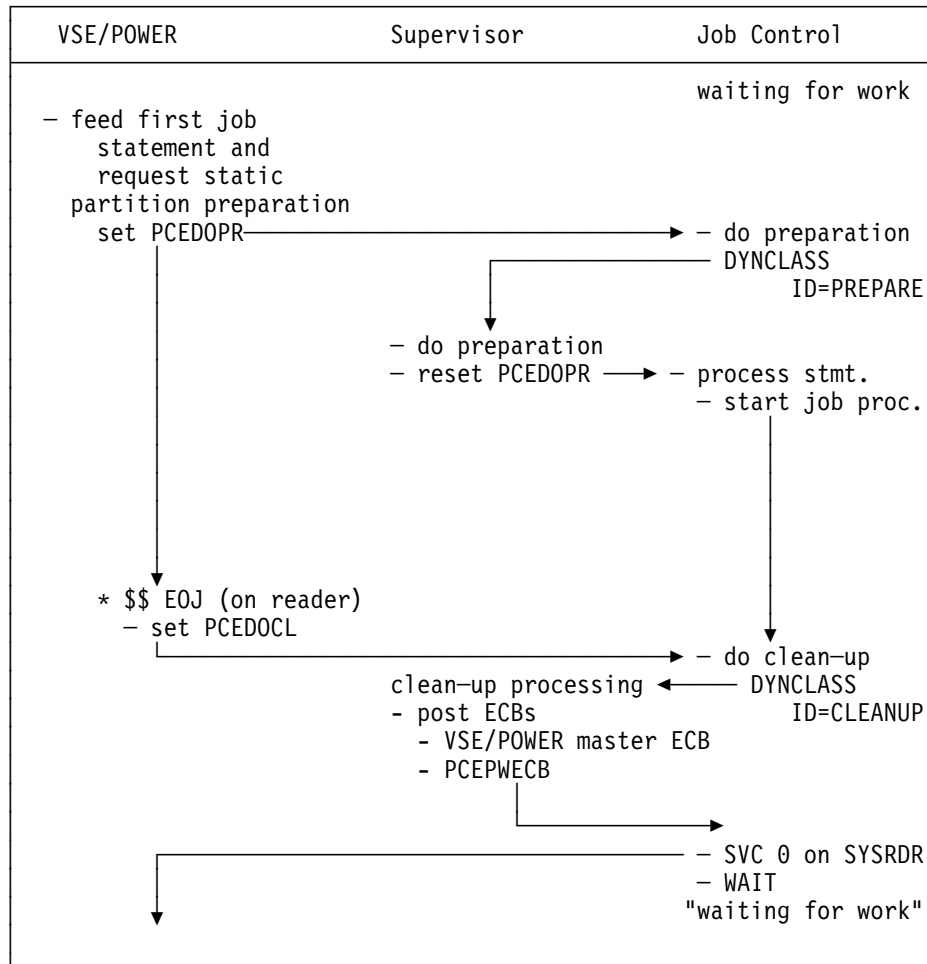


Figure 36. Internals - Static Partition Preparation and Cleanup

## Dynamic Partition Preparation and Cleanup

Before the first z/VSE Job Control statement will be processed, partition initialization is done, that is

- the profile (procedure) specified for the dynamic class (defined in dynamic class table) will be executed by z/VSE Job Control,
- assignments to VSE/POWER spooled devices will be established (as defined by profile).

The preparation is part of the partition initialization. At end of a VSE/POWER job partition deactivation is done, that is VSE/AF Job Control calls:

1. The cleanup service, which undoes assignments, drops LIBDEFs, etc.
2. Unbatch process (including TSTOP).

## Normal Processing

1. Preparation will execute at the beginning of a VSE/POWER job, cleanup will execute at the end of each VSE/POWER job for which preparation was done.
2. Preparation/cleanup (P/C) will not execute in the VSE/POWER partition instead it will run in the VSE/POWER-controlled partition being prepared/cleaned up. Preparation will be called after start of the partition before Job Control is

- loaded. Job Control will call Cleanup. P/C is considered to be part of the user's job.
3. VSE/POWER posts Job Control for cleanup. "Do cleanup" will be indicated in the PCE control block. The indication must not be reset.
  4. For dynamic partitions no ASI procedure will be executed, instead a profile (procedure) has to run after preparation. The same mechanism as for ASI processing will be used.
  5. For dynamic partitions the profile has to contain assignments for SYSIN, SYSPCH and SYSLST. Initially SYSLST and SYSPCH are assigned IGN.
  6. When VSE/POWER receives control after the first SVC 0 to a VSE/POWER spooled reader, VSE/POWER resets PCEINIT in PCEFLAG.
  7. From VSE/POWER's point of view, deactivation for dynamic partitions will include the UNBATCH logic with respect to freeing resources held by the partition (for example, GETVIS, assigns, LIBDEFs, etc.).
  8. After deactivation is complete the ECB in the PCE control block is posted to inform VSE/POWER.
  9. VSE/POWER deallocates (ALLOC) the dynamic partition.

### Cancel Conditions

- For the first read request (to get the first job statement) during initialization (PCEINIT in PCEFLAG) SYSIN has to be assigned, otherwise Job Control cancels the partition.
- IJBSINP calls the service routine TSRICNCL, when no dynamic space GETVIS area is available for the first call of IJBSINP (in end-of-task processing) during initialization.
- When a dynamic partition is canceled during Initialization,
  1. the cleanup routine is called during end-of-task processing,
  2. PCEDOCL and PCEICNCL flag is set,
  3. job control is loaded and because PCEDOCL is set, processes unbatch,
  4. TSTOP processing posts the VSE/POWER partition (PCEPWECB),
  5. the whole VSE/POWER job stream is flushed (up to \* \$\$ EOJ),
  6. VSE/POWER deallocates the partition.
- When a VSE/POWER job executing in a dynamic partition is canceled after profile processing, the system behaves the same as if a static partition was canceled.
- If VSE/POWER has to cancel (TREADY) a controlled partition due to VSE/POWER problems (for example, Data File I/O error or cancel of VSE/POWER itself), cleanup will be done automatically during maintask EOT processing of the canceled partition. Thereafter a dynamic partition is deactivated by job control (see above) and deallocated (ALLOC) by VSE/POWER. Note that VSE/POWER is posted via the PCE control block ECB.

Action to be taken	Initiated by	Function to be called
1) Job submission to VSE/POWER	II	
2) Request a partition – Allocation of dynamic partition	VSE/POWER	ALLOC
3) PSTART the partition	VSE/POWER	
4) Partition initialization	Superv.	
5) Schedule the job in this partition	VSE/POWER	
6) VSE/POWER job is in execution		
7) * \$\$ EOJ reached		
8) Request partition deactivation	VSE/POWER	
9) Free all allocated resources of this VSE/POWER job – initiate clean up – include logic of the JC UNBATCH statement, do deactivation (TSTOP), post VSE/POWER)	Job Cont.  superv.	DYNCLASS ID=CLEANUP  TSTOP COND=UNBATCH
10) Reset spool indications	VSE/POWER	
11) Request partition deallocation	VSE/POWER	ALLOC

Figure 37. Scenario: Dynamic Partition Preparation and Cleanup

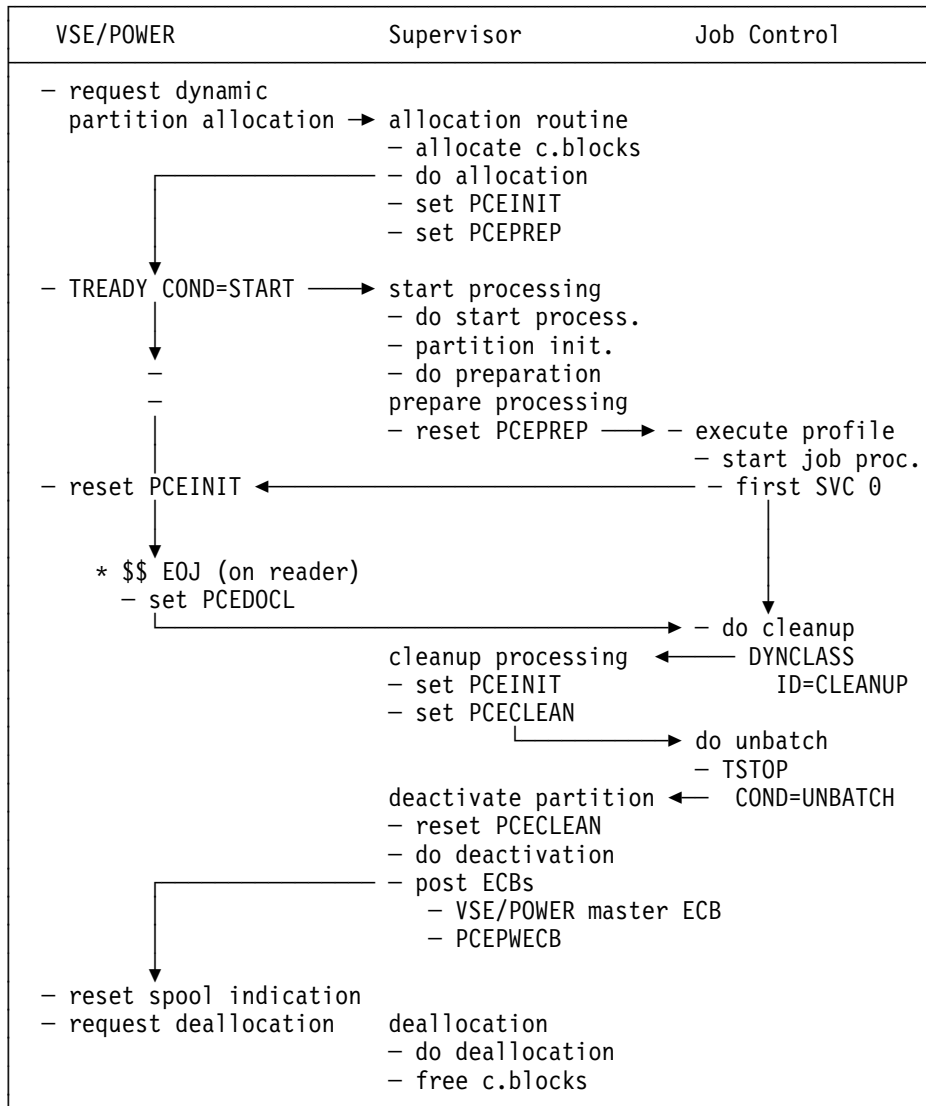


Figure 38. Internals - Dynamic Allocation, Initialization, Deactivation and Deallocation



---

## VSE/AF Dispatcher - Identifiers and Limits

### Number of Partitions, Task and Partition Key Definitions

#### Number of Partitions

The current design limit for z/VSE is 212 partitions, which results from the 255 VSE tasks. Each VSE partition occupies at least one VSE task (maintask). 32 VSE tasks are reserved for system use (system tasks including attention routine task). 11 VSE tasks are reserved for system functions and VSE/POWER basic subtask requirements.

====> Design Limit 212 = 255 - 32 - 11

which allows a maximum of 200 dynamic partitions and 12 static partitions.

The maximum number of partitions in a system can be calculated as follows:

Maximum Number of Partitions = 255

- 32 system tasks (incl. AR task)
- subtasks for z/VSE components
- subtasks for user applications

However the maximum reasonable number of concurrent allocated partitions is dependent on:

- the CPU,
- the system configuration,
- the customer environment and
- the job profile.

#### Notes:

1. The maximum number of partitions within the system can be specified via the z/VSE IPL SYS command (NPARTS parameter). The generated default is 12 static partitions (no dynamic partitions).
2. Maintasks for the specified partitions (via NPARTS parameter) are reserved during IPL.

#### Storage Protection Key

Each static partition in VSE is assigned a unique storage protection key, dynamic partitions have always the same storage protection key. A storage protection key is the hexadecimal representation of the value  $16^n$ , where

$n$  is the partition id of static partition

Dynamic partitions use the storage key  $16^{13}$  (may be changed in future). Storage protection keys are assigned according to the scheme shown in Figure 39 on page 94, where partition IDs X'0D' to X'11' show allocated dynamic partitions of dynamic class X (X1, X2, X3) and Y (Y1).

Part. id	Part. name	PIK Value in COMREG	Storage Key
00	SYS	0000	00
01	BG	0010	10
02	FB	0020	20
03	FA	0030	30
04	F9	0040	40
05	F8	0050	50
06	F7	0060	60
07	F6	0070	70
08	F5	0080	80
09	F4	0090	90
0A	F3	00A0	A0

Part. id	Part. name	PIK Value in COMREG	Storage Key
0B	F2	00B0	B0
0C	F1	00C0	C0
0D	X1	00D0	D0
0E	X2	00E0	D0
0F	Y1	00F0	D0
10	X3	0100	D0
11	Y2	0110	D0
12	*	0120	D0
...	...	...	...
D4	**	0D40	D0

Figure 39. Partition Identification and Storage Protection Key

**Notes:**

- \* No dynamic partition is allocated to the PIK value of X'0120', therefore no partition name (SYSLOG id) is available.
- \*\* Shows the highest possible entry in the PIK table which can be defined via the z/VSE IPL SYS command, NPARTS parameter. In our example the entry is unused, that is no dynamic partition is allocated to this entry.

**Partition Identification**

A partition is identified by its unique 2 byte Partition Identification Key (PIK). In some cases a partition is identified by a 'Partition Identifier' (PID) value which is just the value PIK/16.

**Task Identification**

Tasks are identified by a 2-byte hexadecimal number.

When "support for more tasks" is not activated on z/VSE 4.2 (using the SYSDEF SYSTEM command), the first byte of this 2-byte number is always x00. This is the same as with z/VSE 4.1 and below. The task identifiers (task-ids) are in the hexadecimal range from x'0001' to x'00FF', which means that 255 concurrently active tasks are supported.

With z/VSE 4.2 the user may request (using the SYSDEF SYSTEM command) that more than 255 tasks can be activated at the same time. The maximum number of concurrently active tasks is 512. The task-ids are in the range from x'0001' to x'0200'.

The following table shows the task identifier (TID) values and their assignments to particular tasks:

System Task		Main Task		Sub Task	
TID		TID		TID	
00	- Reserved	20	AR	n	**
01	SNS - Unit check processing	21	BG	n+1	**
02	DSK - Resident disk error handler	22	F1	n+2	**
03	RAS - Channel/machine check handler	23	F2	n+3	**
04	PMR - Page manager	24	F3	n+4	**
05	- Reserved	25	F4	n+5	**
06	PGN - Page-in processing	26	F5	...	
07	SUP - Program Fetch processing	27	F6	n+m-1	**
08	DIR - Directory read processing	28	F7		
09	VTA - Virtual Tape Processing	29	F8		
0A	AOM - Asynchronous Operations Proc.	2A	F9		
0B	ERP - Transient error recovery	2B	FA		
0C	LCK - Lock-Unlock processing	2C	FB		
0D	CMT - Capacity Measurement Task	...	*		
0E	LOG - Access control processing	...	*		
0F	SVT - Special service processing	...	*		
10	DSP - Dispatcher system task	...	*		
11	SPT - Service processor				
12	CST - Console support task				
13	HCF Hard copy file task				
14	FCP SCSI processing				
...	up to 10 vendor system tasks				
1F	...				
20	AR - Operator communication				

Figure 40. Task Identifier (TID) Values

**Notes:**

- \* Depending on the number of partitions up to 212 maintask TIDs may be used.
- \*\* A pool of subtasks is created and maintained by the supervisor. The size of this pool is given by the maximum number of subtasks active at the same time, where
   
 $x = \text{highest maintask TID} = X'20' + \text{number of partitions (SYS NPARTS=...)}$ 
  
 $n = \text{max.}(X'30', x + 1)$ 
  
 $m = \text{MTASKS} - n = \text{maximum number of subtasks}$ 
  
 where MTASKS is either 256 (if the support for more tasks is not active) or the NTASKS value from the SYSDEF SYSTEM command.

**Identification of Current Partition and of Current Service Owner**

Before control is given to a task the dispatcher sets up the PIK field (bytes 46-47 of the background communication region - BG-COMREG) by a partition identifier key value.

In case of a system task it is the PIK value of the service owner partition (in special situations it may be the system partition key). In case of a user task it is the PIK value of the task's home partition.

**Note:** Whenever a task of the BG partition is active, the PIK field is set to the partition identifier key of the BG partition. Since bytes 46-47 of the other communication regions are generated with the corresponding foreground partition identification keys, any active user task may find its own partition identification key via its own COMREG.

## Identification of Current Task

Before the dispatcher gives control to a task, it puts the task identifier into the TID field at displacement 90-91 in the system communication regions (SYSCOM). The TID value in the TID field identifies the task which is currently active. This may be any system or user task.

## LTID (Logical Transient Owner)

The LTID, a halfword (LIK) at displacement 88 in the system communication region (SYSCOM) contains the same value as the TID when the Logical Transient Area (LTA) is in use and, therefore, identifies the owner of the LTA. When the LTA is free, the LTID is zero. The SVC 2 (X'02') routine sets the LTID, and the SVC 11 (X'0B') routine resets it to zero.

### Notes:

1. Do not use this interface anymore.
2. Any logical transient routine may find its own task identifier by using the TID field.

## LTK (Logical Transient Key)

The logical transient key, a halfword (LTK) at displacement 110 in each partition communication region (COMREG). In a foreground communication, the key value in the LTK is not significant. The LTK in the background communication region (BG-COMREG) has the same value as the PIK of the partition of the task that owns the LTA, or contains zeros when the LTA is free. When the LTA is occupied by the task, therefore, the BG-COMREG has the same value in its LTK as in its PIK when the owning task is active.

**Note:** This LTK interface should not be used anymore.

---

## Physical Input/Output Control System (PIOCS)

Physical IOCS is that portion of the resident supervisor that:

- Builds a schedule of I/O operations for all devices on the system (CHANQ Table).
- Starts the actual I/O operations on a device (SSCH Instruction).
- Monitors all events associated with I/O operations.
- Performs error recovery actions. Refer also to *z/VSE Supervisor Diagnosis Reference Error Recovery and Recording Transients*, SC33-6326 and *z/VSE Supervisor Diagnosis Reference Logical Transients and \$IJSxxx Phases*, SC33-6324.

---

### I/O Request Enqueuer

When a channel program is to be executed for a user, the I/O Request Enqueuer routine first checks to see if a channel queue entry is available.

If the channel queue is full, the issuer is set CHANQ-BOUND until a channel queue entry is available again, which is normally the case after completion of I/O interrupt processing.

**Note:** The occurrence of this bound condition is an indication that the number of CHANQ entries, either the default value or the value specified at IPL time, is less than the number of concurrent I/O requests. Low performance may be the result. This situation could have been prevented by either specifying, or increasing the specified CHANQ value of the SYS-command at IPL time or by inactivating some of the TP-devices assuming that the maximum number of CHANQ entries had already been specified.

If an entry is available in the channel queue, the GETPUB routine first validates the users parameters and checks them for correctness (Error Exits: ERR21, ERR25, ERR26, ERR27). In case the users input has been proven to be correct, the I/O request enqueuer does some special work for special devices and/or components.

- For all I/O requests directed to a device which is logically assigned IGN (Ignore):

It ensures that these types of request are immediately posted I/O complete without having actually been started.

- For all I/O requests being issued from within a VSE/POWER controlled partition and directed to an UR device which the user did define as 'spooled' device or directed to the operator console:

It ensures that these type of requests are being passed to the VSE/POWER SVC 0 appendage, and will be further processed due to the information returned from that routine.

- For I/O requests directed to the system operator console (SYSOCDEV):

It ensures that these requests are passed to the Console Router (described in the Console Functions DRM) for further processing.

- For DASD and Diskette I/O requests issued from user tasks:

It ensures that the associated channel program starts with a command considered valid by the VSE system (ERR33).

(Refer also to system files described later in this section.)

It ensures proper DASD file protection in case the user specified DASDFP=YES (ERR42).

Special processing information is saved in general register 5 until a CHANQ entry has been allocated (after CCW Translation).

If the I/O request needs to be translated control is passed to the CCW-Translation Routine (described later in this chapter) to get the virtual channel program copied into the copy blocks within the supervisor and to get all virtual addresses translated to their correct real addresses.

The CCW-Fixing Routine is used to get all referenced I/O areas TFIXed, if they are not already PFIXed, thus making sure, that this page will not be paged out by the PAGE MANAGER routine.

In the next steps, all the information which is needed to further process any I/O operation is saved in the CHANQ entry which is then enqueued into the chain of I/O requests that might already be waiting for this device. The request is enqueued due to its I/O priority retrieved from the I/O priority table (HQUPRI and HQUPPRI). (For a sample of SYSIO request enqueueing refer to Appendix C, "Samples" on page 513.

If the just enqueued I/O request is NOT THE FIRST one in the device chain, control goes directly back to task selection.

If the request IS THE FIRST one in the device chain, the CPU time is charged to the partition which issued the I/O request (refer to Job Accounting described later in this chapter) and control is passed to the Device Scheduler Routine.

Special processing support provided by the I/O scheduler or related SVC-routines will be described on the following pages.

## Block Protection (SVC 35)

Block protection ensures that a 'block' on a disk device which is being held by one task is not accessed by another task unless the holding task has released the 'block' again.

*CKD Devices (BBCCHH):* The unit of protection is one track. The track address is retrieved from the users SEEK CCW, which must be the first CCW. The whole track is always protected against access by another task.

*FBA Devices:* The unit of protection is always the range of FBA blocks as specified in the DEFINE EXTENT CCW which must be the first CCW. The whole range of blocks is protected against access by another task.

If the first CCW is not a SEEK or a DEFINE EXTENT CCW, block protection is simply ignored and normal SVC 0 processing is done. All requests to protect a track on a CKD device or a range of FBA blocks against simultaneous use will be entered into the Track Hold Table before the I/O Request Enqueueer gains control. The block protection routine forces the issuing task to be set TRK-bound if the given block is already held by another task. It will be reactivated as soon as the requested block becomes available which is normally the case after the holding task has released the track.

Multiple I/O requests for tracks or ranges of FBA blocks which are to be held are chained in a device chain with forward and backward pointers, and the appropriate PUB contains the index to the first Track Hold Table ENTRY. For the format of the Track Hold Table (THTAB) see Figure 233 on page 512.

## System Files

The SYSFIL support of the supervisor allows to have system files (SYSRDR, SYSIPT, SYSPCH and SYSLST) on CKD and/or on FBA disk devices. The scheduler turns on a special bit in the CHANQ entry to ensure proper program flow within the I/O supervisor. Special processing however, is required for system files residing on FBA devices.

*System Files on FBA Devices:* SVC 103 (X'67') performs the input/output operations for system files on FBA devices. The code of the SVC 103 (X'67') consists of:

- The resident part, performing supervisor functions.
- The pageable part, loaded into the SVA, performing data management (blocking/deblocking) functions.

For details see description of SVC 103 (X'67').

---

## Scheduler

The Scheduler is entered from either the I/O Request Enqueuer to drive a single device, or it is entered from the I/O Interrupt Handler (described later in this section) to get the next request in a queue (if any) started. The Scheduler ensure that all requests which have been enqueued by the I/O Request Enqueuer are started in FIFO order as soon as the resource (Channel, Subchannel or Device) is, or becomes available. The Scheduler ensures the accessibility of a device. In case the device is gated, control is directly passed to task selection. If the device is available, the Scheduler does some SIO-preprocessing for special devices.

- For SYSIN I/O requests:  
It ensures not reading past /& (ERR30)
- For Tape I/O requests:  
It ensures control to be passed to the Tape ERP whenever Tape ERP has previously requested that the next SIO attempt is to be routed to Tape ERP. The actual user request will be delayed until Tape ERP has completed its pre-processing function.

In case of an I/O error on a previously initiated I/O operation it ensures the recovery channel program, as specified by the ERP System Task and not the channel program as specified in the user's CCB to be initiated.

It ensures that Tape I/O operations which do not meet the above criteria are being suspended for normal processing until the service (SVT) task has completed its eventually ongoing Tape volume recognition process.

All other tape requests, not meeting one of the conditions described above are ensured to be started with the assigned (PUB) Mode setting.

- For Advanced Function Printer (AFP) I/O requests:  
It ensures control to be passed to the AFP-ERP whenever AFP-ERP has previously requested that the next SIO attempt is to be routed to Tape ERP.
- For DASD-Devices:

The I/O Scheduler ensures that the user can only access those Records or EXTENTS on a volume, that he is authorized to access (ERR30, ERR32).

Following the SIO-preprocessing the I/O Scheduler actually carries out the requested I/O operation by means of a

SSCH

instruction. Depending on the resulting condition code the Scheduler either enters the I/O INTERRUPT PROCESSOR to further process condition codes 1 (CSW STORED) and 3 (DEVICE NOT OPERATIONAL) or it completes its Device Scheduling process by updating the appropriate SIO processing and SIO accounting information and passing control to the task selection routine.



---

## I/O Interrupt Handler

The I/O interrupt handler is entered when an I/O interrupt occurs, or whenever the I/O New PSW (FIXED STORAGE LOCATION X'78') is being loaded. The code to handle I/O interruptions is in the IOINTER part of the supervisor. The I/O interrupt handler will first issue a TSCH (Test SubChannel) instruction to retrieve the status information into an IRB (Interrupt Response Block). The information will then be re-mapped into the /370 fixed storage locations (CSW, ECSW and X'BA') to maintain subsystem compatibility.

An I/O interruption occurs when an I/O operation terminates or the operator intervenes on the device. The interrupt parameter from low storage location X'BC' is used to allocate the PUB entry and to set up the related I/O pointers. It should be noted here, that, in order to prevent system hangs, a PUB *must* have been defined for any device of the installation, regardless of whether this device is being used or not. Before an I/O interrupt for a known PUB is actually processed, privileged components (VTAM, POWER) are given the ability to inspect the Channel Status Word (CSW) via a BAL-type interface (Channel End Appendage (CEA)).

If none of the above conditions exists, the CSW is examined and action is taken according to the table in Figure 41 on page 102.

CSW Status Bit On	Status Condition	Action
45 46	Channel Control Check Interface Control Check	Branch to the Channel Check Handler to interrogate the bits attempting recovery.
38	Unit Check	Retrieve the sense information from the device and if user wants sense data, provide it. If user did provide its own error routine, or if the issuing task was a system task, post I/O request complete with error.
42 43	Program Check Protection Check	If user did provide its own error routine, or if the issuing task was a system task, post I/O request complete with error.
44 47	Channel Data Check Channel Chaining Check	Retry the I/O operation several times and if still persisting, treat as error.
32	Attention	For attention from the operator console (SYSOCDEV) activate the CST-System task to further process this request. Branch to task selection routine.  Attention interruptions from other devices are not processed directly.
35	Busy	Indicate that the device is to be restarted. Branch to General Exit routine.
36	Channel End	Post user
37 34	Device End Control Unit End	Post user and/or reschedule the device. Restart the device.

Figure 41. CSW Testing in I/O Interrupt Handler

If the device status indicates that the channel program has been completed, the appropriate information is posted in the CCB/IORB.

If the CCB/IORB indicates that this is a copied CCB/IORB (X'20' in byte 6) control is given to a special routine (CSWTRANS) which,

- Frees all pages fixed for I/O areas.
- Retranslates the CCW address placed in the copied CCB/IORB to the correct virtual address.
- Releases the CCW copy blocks and IDAL blocks.
- Moves changed parts of the CCB to the virtual-mode program and release the CCB copy block. If the virtual CCB is not in real storage, the end of channel information is not copied to the virtual CCB by CSWTRANS. Instead, CSWTRANS posts a bit in the corresponding task information block (TIB) indicating to the dispatcher that the CCB should be moved before the task is dispatched. This is necessary because no page faults are accepted during the IOINTER process and the page fault will thus be delayed and handled under users task id, next time he is dispatched.
- Activate tasks waiting for copy blocks or waiting for page frames.

CSWTRANS returns control to the interrupt handler when it has finished processing. The I/O Interrupt Handler will then dequeue the CHANQ entry from the channel queue, assuming this was the final interrupt for a specific request and pass control to the Channel Scheduler to get another or the same device started again.

## Automatic Volume Recognition (AVR)

This facility keeps track of device-specific information of each DASD and TAPE device in the system. The supervisor maintains a table, the volume characteristics table (VCT), which contains the specific information for each device (see "Layout of the VCT and DCT Tables" on page 105).

SVC 99 (GETVCE macro, see also Appendix B) retrieves data from this table for the user. The Service System Task (SVT) facility interrogates the device when requested, and updates the table. Requests for updating the table are made by the I/O Interrupt Handler whenever the device becomes 'READY' and by SVC 101 (MODVCE macro, see also Appendix B), which can be issued by any user, but especially IPL, utility programs when a change to the device is suspected and by command processors (e.g. ONLINE and VOLUME command). The request flow is shown in Figure 42 on page 104.

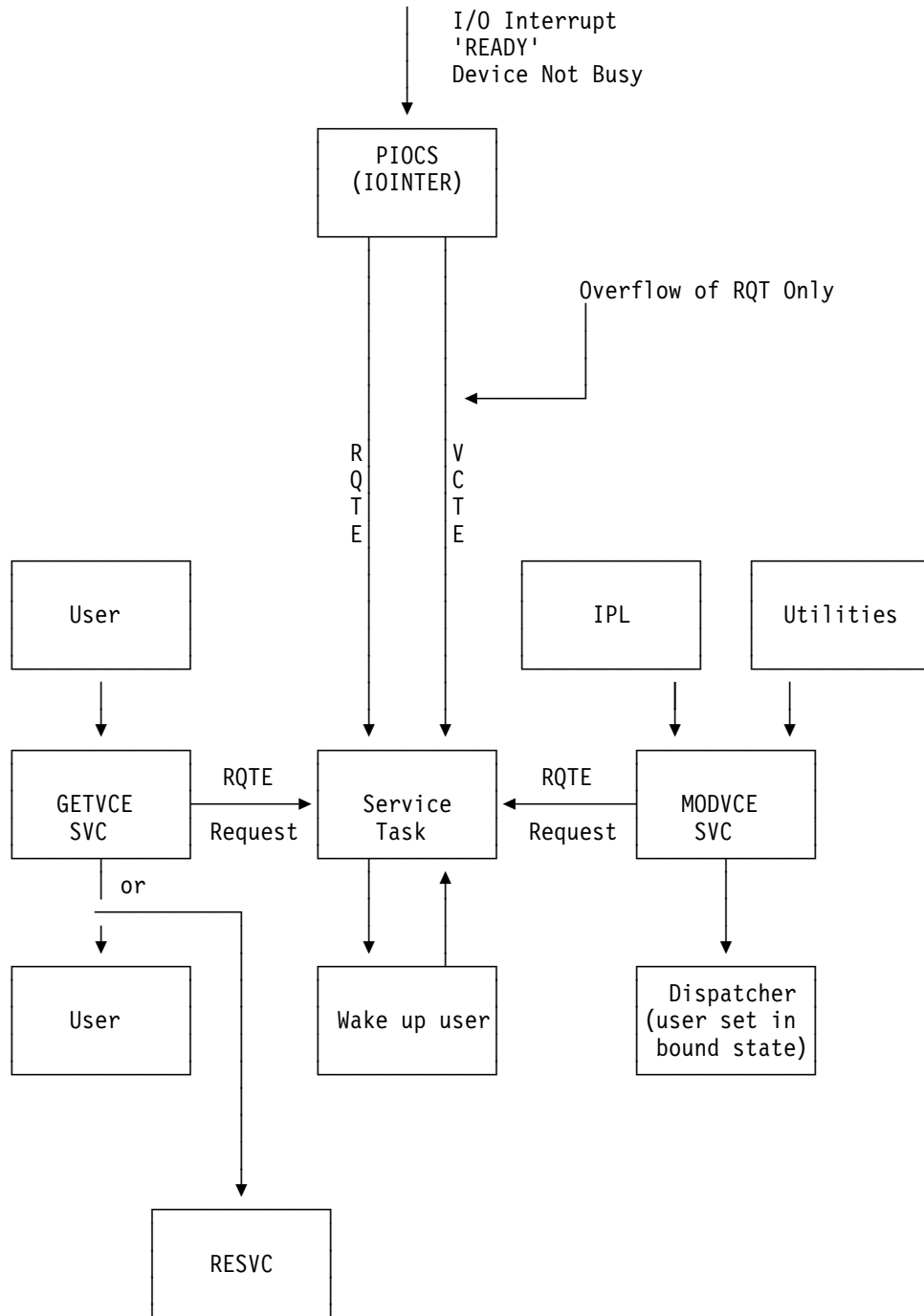


Figure 42. General Flow of Volume Characteristic Table Entry Update

## Updating the VCT Table

The update request may come from two sources:

1. From the I/O INTERRUPT PROCESSOR whenever a DASD- or TAPE-Device became 'READY'
2. From SVC 101 (MODVCE).

The first category must issue the request immediately, since it cannot save the status. The second category, however, requires that the requesting task is readied not before the VCT-Table has been updated. Also, more than one task can request an update for a given device at a time, and if an entry is in the process of being updated, any GETVCE request must be queued in order to wake up the requesting task after the update. This results in two request queues:

- Device-related only.
- Task- and device-related.

The first queue is the VCT table itself. Each entry (VCTE) has a work-to-do and work-in-progress flag.

The second queue is the RQT request table, a simple vector of task IDs, PUB indexes, and function flags. The address of the RQT table can be found at label RQTTAB. If the RQT overflows, the SVC can be retried at a later time using RESVC.

### ***Layout of the VCT and DCT Tables***

The Volume Characteristics Table (VCT) entry contains mainly the information that is defined by the AVRLIST macro (see Appendix B). The Device Characteristics Table (DCT) entry is described by the DCTENTRY macro. Each entry is fixed length and describes the device characteristics of a CKD, FBA DASD or TAPE device. The DCT entry is immediately appended to the end of each AVR entry of the appropriate device.

Usage of the two macros (AVRLIST and DCTENTRY) is discussed under "GETVCE Macro" in Appendix B. The start address of the VCT table can be found at label AAVRTAB.

## Control Unit Initiated Reconfiguration (Quiesce/Resume support)

This part of the supervisor supports control unit initiated reconfiguration.

***The CUIR Architecture:*** Basically the processing of a request is done in the following steps:

- The control unit sends an attention interrupt to the host system.
- The system sets up CCWs to read all pending attention messages. It then checks if the attention message is for quiesce resume purposes. VSE does not interpret any other kind of attention messages.
- The system identifies the scope of the request. This scope is the set of device/path (subchannel/chpid) pairs that are affected by the request.
- The system tries to apply the requested operation to the devices/paths in the scope (i.e. either quiesce or resume).
- It sends back a response code to the Control unit indicating success or reason of failure.

This processing is accomplished by the VSE service task. The service task is informed about any incoming attention interrupt from a device that could support quiesce resume. It then enters phase \$IJBCUIR in the SVA to do the processing. If no further processing is required it unposts itself again.

**The phase \$IJBCUIR** All the processing of quiesce resume takes place in the phase \$IJBCUIR in amode 31. The phase consists of three parts, the code, a stack area and a wrap around debug buffer. All the data required in the processing are kept in an area named QRSGLOB, which is generated in the system task code in the supervisor. Both the supervisor part and the code in \$IJBCUIR are generated by macro SGCUIR. Processing starts in routine QRBASE, which is basically a loop scanning all PUBX entries of the system for attention interrupts. Whenever an attention interrupt is found it reads the attention message. If the attention message is a quiesce resume request, routine QRMAIN is called to do the processing. QRBASE will continue reading attention messages until the device indicates "no more messages available". Scanning for new attention interrupts will only stop, if the PUBX was scanned completely and no indication of an attention interrupt was left.

QRMAIN first calls a subroutine to determine the scope of the request. This scope is encoded in the reconfiguration request record and can not be seen directly from it. The system has to read a so called configuration data record from every chpid in every subchannel that could be connected to the same control unit. The reconfiguration request record contains an NED-map, an NEQ-map and a SELECTORBYTE identifying the set of configuration data records that come from the subchannel/ chpid pairs making up the scope of the request. Each configuration data record consists of 32 byte entries. There is three different types of entries:

- NEDs, node element descriptors. Each NED represents one separately identifiable component in a subsystem and contains its serial number and further information. Only those NEDs that are flagged as "reconfiguration NEDs" are required for the CUIR processing. One of the NEDs of a configuration data record is flagged as so called "token NED".
- NEQs, node element qualifiers. These entries are not used for reconfiguration request processing.
- GNEQ, the general node element qualifier. It should be the last entry in a configuration data record. It contains a 24 byte area which is required in quiesce resume processing.

The system first needs to find the so called base configuration data record. In order to do this, the system first reads the configuration data record from the same subchannel/chpid pair that it received the attention interrupt from. If the selector byte in the reconfiguration request record is zero, this configuration data record is the base configuration data record. If it is not zero, the system has to scan all configuration data records that have the same token NED as the one that was read from the same subchannel/chpid pair that it received the attention interrupt from. The first configuration data record having the same selector byte value in its GNEQ is the base configuration data record.

When the base configuration data record is found the system will determine the set of related configuration data records. For this it will check each configuration data record having the same token NED as the base configuration data record. If those parts of a configuration data record, that are indicated in the NED map and the

NEQ map of the reconfiguration request record, are equal to the corresponding parts in the base configuration data record, the record is a related one.

Each bit in the NED map identifies one of the so called reconfiguration NEDs of a configuration data record. Bit 0 the first one, bit 1 the second and so forth. The configuration data record needs to be scanned, to find the NEDs in question and compare them to the corresponding ones in the base configuration data record. Each of the bits in the NEQ map refers to a byte in a 24 byte area in the GNEQ of the base configuration data record. The contents of these bytes must be similar to the contents in the GNEQ of the examined configuration data record to make it a related one. All the subchannel/chpid pairs that have been used to read the related configuration data records make up the scope of the request.

When the scope is determined it is handed back to the main routine in a list. This list is processed from the first to the last entry. If processing stops due to some error condition being found there is no recovery for the actions already taken. If for example a list of paths is presented, that need to be quiesced and the list contains five entries and processing encounters an error in the third entry, the first two paths will remain quiesced, whereas entry three to five are not processed.

**Messages issued by CUIR processing:** Since the quiesce resume processing is an infrequently used part of the system, there are messages issued for almost every step the system takes. The messages range from 1H40I to 1H60I. The normal sequence of messages on the console should be as follows:

<b>1H40I</b>	quiesce resume message was received. The message is displayed for every reconfiguration request record that was read. The data displayed are the first part of the record.
<b>1H4xI</b>	indicating an action. Either a device or path was quiesced, or a device or path was resumed. In case of a 1H52I a system identification was sent to the requesting control unit.
<b>1H54I</b>	indicating the response that was sent back to the control unit. If the response was positive or not can either be seen from the sequence of messages 1H4xI, or from the first byte of the response data. The following values are defined by the architecture:
<b>x01</b>	Request was completed successfully
<b>x02</b>	Request rejected. Quiesce resume processing not supported.
<b>x03</b>	Invalid reconfiguration request.
<b>x04</b>	Operator denied request.
<b>x05</b>	Reject of quiesce path request, last path affected.
<b>x06</b>	Reject of quiesce device, device is in use.
<b>x07</b>	Reconfiguration failed during vary processing
<b>x08</b>	Reconfiguration failed due to program failure.
<b>x09</b>	Unknown request type received.

Aside from these messages, there is a number of messages for unusual conditions found by the system. These messages contain data that are intended for debugging

purposes by an IBM support team. All quiesce resume messages are produced by the message writing routine QRMSGWRT in macro (SGCUIR). The following list gives a description of the data that are just indicated by x.xx in the message manual. xn(a-b) refers to the bytes a-b of the printed variable number n in the message. If the message reads 1H55I QRES RESPONSE x.xx REJECT DATA x.xx , the part behind the word RESPONSE is referred to as x1 and the x.xx data behind the word DATA as x2.

- 1H55I** The service task received a disaster error on the attempt to send an answer back to the requesting control unit. x1(0-9) is the answer that should be sent to the control unit, beginning with the return code. x2(0-1) is the cuu, x2(3) the chpid, x2(3-11) the ccb, x2(12-20) sense bytes 0-8 and x2(21-26) sense bytes 22-27 of the failing operation.
- 1H56I** An io error occurred during read of the attention message. x1(0-1) is the cuu, x1(3) the chpid, x1(3-11) the ccb, x1(12-20) sense bytes 0-8 and x1(21-26) sense bytes 22-27 of the failing operation.
- 1H57I** System received an attention interrupt on a path, on which we cannot issue a read for the attention message since the IO on this path is prohibited by some disabling path mask. x1 is the path mask x2 the cuu we received the attention interrupt on. x3(0-7) are the PUBX fields PBXNOERP, PBXNOPVF, PBXNOOPR, PBXNOQSD, PBXPIM, PBXPAM and PBXLPM. x3(8-15) are the chpids in PBXCHPID.
- 1H58I** System received an invalid configuration data record. x1 is the path mask, x2 the cuu on which we received the record. x3(0) is set to 6 if the cdr does not contain a token NED and 7 if the last 32 byte entry in the cdr is not a GENERAL NEQ. x3(1-4) is the storage address of where the cdr is kept in storage and the rest of the data is the beginning of the CDR. This last part can only be used to see if these data make sense as a cdr at all.
- 1H59I** We received a bad return code from a getvis invocation.
- 1H60I** System has encountered an internal processing error. x1 is the field QRERRIND in the SGCUIR code. It should have one of the following values:
- 1** System could not identify a base configuration data record. None of the records tested had the required selector byte in its GNEQ. Set by subroutine QRGSCOPE.
  - 2** There is a bit in the NED map set to on, but the configuration data record currently scanned does not have a corresponding reconfiguration NED. Set by subroutine QRGSCOPE.
  - 3** We could not obtain sufficient GETVIS storage to build the scope list which is scanned by QRMAIN. Issued by QRGSCOPE.
  - 4** Same as 2, just detected in a different place in the code.



**Debugging Aids.:** There is a number of debugging aids incorporated in the SGCUIR code. First the data area QRSGLOB can be located by the eyecatcher preceeding it. The first 3 pointers in it help in finding the \$IJBCUIR phase in storage. Part of that code is a wrap around debug area, the format of the entries of which is described in the QRSTRACE subroutine in SGCUIR. It contains an entry for each vital processing step of a reconfiguration request.

If the code encounters a severe logic error, it does not hardwait the system but leaves to the SGSER service task code. In addition it zeroes the pointer to \$IJBCUIR in the SVA address vector table. This ensures that the code is not entered any more. A dump can be taken any time from then on and should still contain all the data from the time the logic error was detected. One possible reason for this exit could be a stack overflow for example. There is no message given on the console to indicate quiesce resume processing was terminated. The fact that it was done can be seen from the fourth pointer in the QRSGLOB area, which is either zero or points to the position in the code where we decided to leave, using BAL R5,QRHARDWT.

For some logic errors that should not occur but are not due to coding errors in SGCUIR an internal error message 1H60I is issued. If the system issues an internal error message, quiesce resume processing is not switched off, but only the current request is terminated.

## System Console Attention Processing

Attention interrupts from the system console device (SYSOCDEV) cause an appendage routine to be called, that posts the CST system task.

---

## I/O Error Processing

When the I/O interrupt handler detects an error condition, it builds an error entry containing information about the error condition and queues it on one of the four chains to further process this error. This section is about the error processing that the four system task error processing tasks do.

*Error Entries:* There is one error entry for each device added at IPL time. This error entry is used for errors related to non-system task requests or to unsolicited interrupts. There is one additional error entry for each system task (except SNS and PGN), which is used for errors related to requests by this system task.

*Error Chains:* One chain of error entries is maintained for each of the four error recovery system tasks: Sense Task (SNS), Disk Error Recovery Task (DSK), Error Recovery Task (ERP) and Machine and Channel Check Handler (RAS). Each error chain consists of an error chain header pointing to the first (if any) error entry in the chain. System task error entries are enqueued on top of the chain, device error entries at the bottom. Any error entry can be in only one error chain at a time.

*General Procedure:* The I/O error processing routine locates the appropriate error entry and removes it, if necessary, from any error chain. After setting the error information, it enqueues the error entry in one of the error chains, depending on the type of error and on the available information. The chain owner is posted, if not already active.

Each chain owner processes its chain in FIFO order. The first entry is dequeued, a recovery action is carried out and the error entry is then passed to another chain, if additional processing is needed; or freed, if the error recovery is completed. Error recovery system tasks always exit to the I/O Interrupt Handler, before resuming operation with the next entry in the chain. When an error recovery task has finished processing the last error entry on the chain, it deactivates itself.

*Sense Task (SNS):* The main function of the sense task is to read the sense data related to a unit check error and to save them, if needed, in the error entry. The error entry is then passed to the disk resident error recovery task (DSK) or to the error recovery task (ERP).

*Disk Error Recovery Task (DSK):* The function of the DSK task is to analyze the sense data related to a unit check error from a disk device and to initiate retries or other recovery actions if appropriate. The error entry is passed to the error recovery task for operator communication and/or error logging, if necessary.

*Error Recovery Task (ERP):* Four distinct functions are assigned to the ERP task:

- Recovery operations for all I/O errors on non-disk devices
- Handling of all operator messages related to I/O errors
- Logging of I/O error information on the recorder file
- Processing of recording requests which are being passed to the ERP via the SVC 44 service.

The activity of the ERP task is monitored by resident code (SGERP). The resident ERP logic decides upon the device type of the failing device whether further processing will be done by the resident tape ERP (SGTAP) or by the transient ERP. The resident tape ERP is managing error recovery for cartridge tape devices. The error entry is passed to SGTAP who performs the necessary recovery actions and

handling of operator messages but not the logging of I/O error information on the recorder file. When error logging is necessary resident tape ERP passes the error entry to the transient ERP.

If transient ERP will do the processing the first error entry from the ERP chain is dequeued and the contents of the error entry are moved into a fixed area (ERQ1), which is accessible to the transient phases (ERBLOC) and which is the only means to get error and process related information from there. The monitor next decides which transient should run first and calls it and passes control to it. The transient (A-transient) may call other A-transients, but will eventually return to the monitor at label ERPEXIT providing information about the action that is to be taken by the monitor.

Other system functions (SVC 44 and the Missing Interrupt Handler) also move information to be recorded directly into the ERQ1 area, when it is not already occupied. In this case, the ERP task first handles the information already available in the ERQ1 area, before processing the next error entry from the ERP chain.

*Machine and Channel Check Handler (RAS):* Functions assigned to the Channel Check RAS task are:

- Logging of I/O error information on the recorder file
- Handling of all operator messages related to I/O errors
- Recovery operations for all I/O errors on non-disk devices

The activity of the RAS task is monitored by resident code. The resident RAS logic dequeues the first error entry from the RAS chain and moves the contents of the error entry into a fixed area (ERPIB), which is accessible to the RAS transients.

## ERBLOC Area

The ERBLOC area is used as interface between the transients involved in I/O error processing and/or Error Recording. The AERBLOC field in the SYSCOM contains a pointer to this area. The layout of the ERBLOC area is shown in Figure 225 on page 499.

## Error Entries

There is one error entry of each device added at IPL time. The pointer to this entry can be found in the PUB extension (PUBX). The device-related error entries varies in its length depending on the number of sense bytes.

There is one additional error entry per system task, except PGN and SNS task. The address to that error entry is contained in the appropriate system task TCB.

Error entries are chained together and enqueued to the appropriate processing task. There is a separate chain for each, the SNS, DSK ERP and RAS task. A bit combination of outstanding recovery operations is used to address the appropriate chain. The anchor address of any of these chains is contained within the ERBLOC area (see Figure 225 on page 499). For the format of the error entries as processed by SNS, DSK, RAS and the ERP see Figure 226 on page 501.

## Loading an ERP Transient

When exit is to be taken to a physical transient ERP system task issues an SVC 5 to get the transient phase loaded into the physical transient area which is then subsequently entered.

To fetch another ERP transient, the active transient phase issues an SVC 5.

---

## Disk Error Recovery

Disk error recovery routines are resident device error recovery routines. They are described below. A-transients (\$\$A) are only fetched when the error is to be recorded, or when an operator message is to be issued.

A-transients are fully described in the *z/VSE Supervisor Diagnosis Reference Error Recovery and Recording Transients*, SC33-6326.

The error recovery actions done by SGDSK are oriented after the actions prescribed in the controller manuals of the DASD devices. For FBA and CKD devices the recovery procedure can be derived from sense bytes 0, 1, 2 and 7. ECKD devices indicate the recovery action in sense byte 25 (Program action code) and messaging and recording needs in sense byte 24.

The messaging and recording actions initiated by SGDSK may in some cases differ from these recommendations. In case a recovery action is prescribed that would exceed the device support normally provided by VSE, this action will not be executed. Instead the error will be treated as unrecoverable.

SGDSK retries failing IO operations on a path selective basis. If an error can not be recovered on a certain path recovery may be done on another path.

SGDSK may be instructed by sense data to fence a path, a storage director or a controller and will do so, using the required diagnostic control commands.

SGDSK will break a duplex pair if the error recovery is exhausted on all paths of the primary device and the nature of the error indicates, that it may be overcome on the secondary device.

SGDSK keeps track of the last recovery actions it performed in a wrap-around debug area (DERDEBUG).

---

## Resident Tape Error Recovery

There are two different tape error recovery routines:

- Resident error recovery routine SGTAP for cartridge tape devices which is described below.
- Transient error recovery for all other tape devices. The A-transients are fully described in the *z/VSE Supervisor Diagnosis Reference Error Recovery and Recording Transients*, SC33-6326.

SGTAP is processing on the ERP chain of error entries that may contain several error entries for one or more devices. These error entries either belong to a PUBX or to a TCB of a system task. SGTAP always dequeues the next error entry from the chain and starts following error recovery procedure for that error condition:

1. Analyze the presented sense data.
2. Assign recovery attributes like recovery action, messaging and recording needs to the error condition. The recovery attributes are leaned upon the actions prescribed in the controller manuals of the tape devices.
3. Perform the recovery action. When recovery I/O has to be done SGTAP is not waiting for I/O completion but initiates messaging for this error condition and starts processing the next error entry queued in the ERP error entry chain.
4. When the recovery I/O request has been completed messaging is done, if still applicable.
5. Recording, if necessary, is initiated by enqueueing an error entry containing all information to be recorded into the ERP error entry chain. Transient tape ERP will later process this error entry asynchronously.
6. If recovery could be completed, SGTAP is left to the ERP monitor SGERP and the next error entry can be processed or a recovery procedure already started can be completed.
7. If a new error condition is presented on a recovery I/O request the old error condition is considered overcome and the new error condition is processed from now on as if it had been the original one.

SGTAP keeps track of the last recovery actions it performed in a wrap-around debug area (TERDEBUG).

---

## ERP Message Writer

The ERP message writer SGEMSG writes all recovery messages with prefix 0P. This is done in following steps:

- Set up message output lines
- Issue message using macro WTO/WTOR
- Analyze operator reply in case of decision-type messages
- Select the proper exit

Usually, the ERP message writer gets either control from the resident tape error recovery routine SGTAP or from the transient error recovery routines. They provide the operator action and target codes by setting up a flag byte in the error entry or ERBLOC area, respectively. The action code may be:

- A (action-type message, operator intervention required)
- D (decision-type message, operator reply required)
- I (information-type message, no operator action or reply required)

For decision-type messages the target code may be:

- I (decide between cancel or ignore)
- IR (decide between cancel, ignore, or retry)
- R (decide between cancel or retry)

For information-type messages the target code may be:

- C (I/O operation will be cancelled)
- I (the task could not be notified about the error)
- R (I/O operation is retried)
- P (the task was notified about the error)

For action-type messages, there is no target code.

---

## Missing Interrupt Handler

The Missing Interrupt Handler (MIH) is a resident supervisor routine that interrogates all entries in the channel queue on an interrupt driven time slice basis. The MIH is entered whenever an ATTENTION interrupt from the system operator console (SYSLOG) is recognized, or whenever the system is going to enter an ENABLED WAIT state.

The MIH will first ensure that a defined time interval has elapsed, otherwise it will immediately return via the linkage register. If the defined interrupt has elapsed, all channel queue entries will be examined to determine whether they have been flagged as long-term entry. If the entry is not a long-term entry, it will be flagged as such if the associated I/O operation has been successfully initiated and if it is a device to be handled (see below). All entries which are already flagged will be further investigated in order to determine why these entries are still in the channel queue, for example, a channel end or device end is outstanding.

For this purpose, any associated I/O interrupt information as well as the current device status, retrieved by means of a TIO instruction, will be used to set up the appropriate message. The result of the TSCH determines whether an information-type message or a decision-type message is provided. For both types of messages, the final action performed by the MIH depends on the communication bytes in the CCB and on the task which issued the I/O operation. All missing interrupts that can be uniquely identified as device errors will result in a record being written to the recorder file in a standard format.

Certain TP devices cannot be supported since the supervisor cannot distinguish between an endless polling loop or a subchannel hanging due to a missing interrupt. These conditions are handled by the individual components, usually by timer interrupts.



# Lock Management

*Locks* a resource against simultaneous use by other tasks.

*Unlocks* a given resource that was previously locked.

The SVC 110 (X'6E') is invoked by the LOCK and UNLOCK macros.

Resources that may be locked/unlocked are:

- Data sets
- Libraries
- Catalogs
- Program routines
- Control blocks, etc.

In a DASD sharing environment the SVC 110 (X'6E') may be used:

- To lock resources against simultaneous use by other tasks of the own system (internal locking), or
- To lock resources against simultaneous use by tracks of another VSE system (cross-system locking).

The SVC 110 (X'6E') routine (the lock manager), including the SVC 63 (X'3F') and SVC 64 (X'40') routines and the associated tables, is contained in the pageable part of the supervisor.

The lock manager is a serially reusable routine. Only one LOCK or UNLOCK request may be executed by the system at a time. If the lock manager is already active, the issuing task will be set to USEBND (X'8B') and afterwards into WAIT state (RESVCX).

## Required Control Information

The resource to be locked/unlocked is described by the control block DTL (Define The Lock), the address of which is passed to the SVC 110 (X'6E') routine in register 1. Register 0 is used as a parameter passing register. The contents of register 0 is used to differentiate between LOCK and UNLOCK.

DEC	Description
0 – 2	Zero
3	Option Flag Byte
	X'80' FAIL=WAITECB
	X'40' UNLOCK JC=SYSID
	X'20' UNLOCK ALL
	X'10' UNLOCK ALL,JC=E0J
	X'08' FAIL=WAITC
	X'04' FAIL=WAIT
	X'02' LOCK (USE) request
	X'01' SVC 110 (X'6E') request

### Notes:

1. LOCK - Option flag byte contains: X'03'
2. UNLOCK - Option flag byte contains: X'01'

Figure 43. Contents of Parameter Passing Register 0

---

## LOCK and UNLOCK (SVC 110 - X'6E')

### Locking a Resource

If a requested resource is available, it is assigned to the requesting task by building an entry for this resource in LOCKTAB and chaining an owner element to the LOCKTAB entry.

If the permanent LOCKTAB resp. owner element space (following the lock manager code) is exhausted, SVA space for LOCKTAB resp. owner element entries will be allocated.

If cross-system locking is requested an entry is placed into the external lock file, too. For the relationship between LOCKTAB and owner elements refer to Figure 191 on page 453.

The SVC 110 (X'6E') routine cannot issue an I/O request to the external lock file. When access to the external lock file is requested, the SVC X'6E' routine changes its status to that of a system task.

If a requested resource is locked by another task of the same system *and* FAIL=WAIT or FAIL=WAITC or FAIL=WAITECB is specified in the LOCK macro, a deadlock test is performed to avoid a soft wait condition. If the system is deadlock free, the requesting task is set into WAIT state (RESVCX) for FAIL=WAIT and FAIL=WAITC. If FAIL=WAITECB the lock request of the issuing task is queued to this resource but control is given back to the caller. It's then the caller's decision where and when he wants to wait until the resource is allocated to him.

A deadlock test is also performed if FAIL=WAIT is specified and the supervisor runs out of LOCKTAB space or of owner element space.

For external locks a deadlock test is performed, if the disk block where an external lock entry should be entered is full *and* all entries of that block are in use by tasks of the own system.

**Note:** Deadlocks, where tasks of different systems lock resources in reversed order, will not be detected.

If a task wants to lock a resource which is locked by a task of another system, the LCK system task sets up a time interval (SVC 10 - X'0A') and sets the requesting task to the "RURBND (X'8E')" state (RESVCX). When the time interval elapses, the timer interrupt handler takes all tasks waiting for externally locked resources out of the WAIT state.

## Lock Options

LOCKOPT	CONTROL	Description
1	E	No other user is allowed to use the resource concurrently.
	S	Other 'S' users are allowed concurrent access, but no concurrent 'E' user is allowed. (Note 1)
2	E	No other 'E' user gets concurrent access, however, other 'S' users can have access to the resource (Note 2)
	S	Other 'S' users can have concurrent access and, in addition, one 'E' user is allowed.
4	E	No other 'E' user from another system is allowed. However, other 'S' users from other systems may use the resource concurrently (LOCKOPT=2 support across systems).
	S	Other 'S' users and in addition one 'E' user from another system is allowed.

**Notes:**

1. Either one 'E' user *or* n 'S' users are allowed (n = number of 'S' users).
2. One 'E' user *and* n 'S' users are allowed.
3.
  - CONTROL=E      Resource is enqueued in exclusive mode.
  - CONTROL=S      Resource is enqueued in shared mode.
  - LOCKOPT=4      Defines a system action, which treats the lock request across systems as a LOCKOPT=2 request.

*Figure 44. Lock Option and Control Parameter*

Incoming LOCK Request		Current LOCK Status of Resource					
		LOCKOPT=1		LOCKOPT=2		LOCKOPT=4	
LOCKOPT	CONTROL	CONTROL=		CONTROL=		CONTROL=	
		E	S	E	S	E	S
1	E	W	W	W	W	W	W
	S	W	G	I	I	I	I
2	E	W	I	W	G	I	I
	S	W	I	G	G	I	I
4	E	W	I	I	I	G/W	G
	S	W	I	I	I	G	G

G = The LOCK request is granted (ret. code = 0).  
 I = Incoming LOCK request is inconsistent with current LOCK status (ret. code = 12).  
 W = Access to resource cannot be granted (ret. code = 4 or 16).  
 G/W = The access is granted, if the resource is already exclusively owned by the own system. The access is denied (ret. code = 4), if the resource is exclusively held by the other system.

Figure 45. System Actions Depending on Control Definition in DTLs

## Unlocking a Resource

When a resource is to be unlocked, the appropriate LOCKTAB entry is cleared to zeros or, if there is more than one user of this resource, the unlocking task is removed from the owner chain of the entry.

If a LOCKTAB entry is cleared to zero, or if the locking status of the particular resource is changed to a lower control level (i. e. from exclusive to shared control), all tasks of the own CPU waiting for this resource are activated so that they retry their lock request. If there is a requestor chain for this locktab entry, the lock manager retries allocation for those tasks after completion of the UNLOCK function.

If a resource is locked 'cross-system' and the locking status is changed, the entry on the external lock file is updated; as a result tasks of another CPU will find the resource available when they retry their lock request.

## **UNLOCK SYSTEM=sys-id (AR-Command)**

All resources, which are held by *another sharing system*, will be freed (unlocked) and the corresponding entries will be removed from the external lock file. 'sys-id' specifies the CPU-id of the other system.

This service can be used only by the Attention task. Any other task issuing this macro, will be canceled with 'illegal SVC'.

### **Return Codes in register 15**

0 (X'00')	Successful request. All locks held by the other system have been unlocked.
4 (X'04')	The specified sys-id has not been found in the external Lock file (the operator specified probably a wrong system-ID).
8 (X'08')	External Lock file damaged.
12 (X'0C')	Irrecoverable I/O error on the Lock file.

## **UNLOCK ALL**

All resources, which were locked by the task with 'KEEP=NO' will be freed (unlocked).

The SVA space of owner elements and LOCKTAB entries (if no more owner elements chained) is released.

UNLOCK ALL will be automatically called at task detach time and EOJ step. When UNLOCK ALL is called during EOT processing and the partition runs in OS/390 emulation mode, additionally the OS/390 ENQ resources obtained by the OS/390 ENQ macro are released.

## **UNLOCK ALL,JC=EOJ**

All resources, which were locked by the issuing task including those with 'KEEP=YES', will be freed (unlocked).

The SVA space of owner elements and LOCKTAB entries (if no more owner elements chained) is released.

At EOJ time (/& or // JOB statement processing) all resources still owned by the partition are freed via UNLOCK ALL,JC=EOJ.

---

## Lock Manager Internals

### Entry Points

SVC110	LOCK / UNLOCK
SVC63	USE
SVC64	RELEASE

### LOCK / UNLOCK Input Registers

Reg. 0	any parameter flags (stored to LOCKPARM)
Reg. 1	DTL address

### Exit

DISP	exit to dispatcher
ERR1E	I/O error on lock file
ERR21	invalid parameter list format
ERR25	invalid parameter list limits
ERR2E	possible deadlock
RESVC or RESVCX	if lock manager in use or resource already locked

### Permanent Usings

Reg. 1	DTL address	(DTLADR)
Reg. 2	LOCKTAB entry pointer	(LOCKADR)
Reg. 6	dispatcher	(DISP)
Reg. A	save area pointer	(SVEARA)
Reg. B	base register	
Reg. C	owner element pointer	(LOKOADR)
	request element pointer	(LOKOADR)
Reg. D	base register	

**Note:** Refer to “Control Blocks related to Lock Management” on page 453.

## Lock Manager Flags

Label	Flag	Description	Value
LOCKPARM		Flag – lock/unlock parameters (in register 0)	
	WAITEFLG	FAIL=WAITECB (request/queue)	X'80'
	CHECKFLG	if only deadlock check active	X'40'
	UNLSYS	UNLOCK JC+SYSID is specified	X'40'
	UNLALL	UNLOCK ALL is specified	X'20'
	UNLEOJ	Request from EOJ routine	X'10'
	WAITCFLG	FAIL=WAITC (conditional)	X'08'
	WAITUFLG	FAIL=WAIT (unconditional)	X'04'
	LOCKSVC	LOCK (SVC110) or USE (SVC63)	X'02'
NEWLOCK	LOCK/UNLOCK (SVC110)	X'01'	
UNLCKFLG	BLKMODF	Flag – unlock SVC (UNLOCK) External block modified (write back)	X'10'
	WAKEUPE1	Activate E1 requestors	X'08'
	FREELE	Give up a LOCKTAB entry	X'04'
	FREEOE	Give up an owner element	X'02'
	WAKEUP	Activate waiting tasks	X'01'
DSHRFLG	LCKSYS	Flag – for lock system task System task is active	X'80'
	LCKTIM	Timer request is already set	X'40'
	LCKREQ	Update on ext. file required	X'20'
	LCKRESVD	Disk drive reserved (lock file)	X'10'

Figure 46. Lock Manager Flags

## Return Codes

### Lock Return Codes

Return Dec	Code Hex	Flag	Description
0	0		Request executed successfully
4	4		Resource owned by other task
8	8	ERRINTSP	LOCKTAB space exhausted
12	C	ERRINCON	Resource request inconsistent with present lock status
16	10	ERRDELO1	Deadlock
20	14	ERRDTLFO	DTL format error
24	18	ERRDELO2	Already locked by issuing task (deadlock)
28	1C	ERREXTSP	Space exhausted on external lock file
32	20	ERRNOVOL	Volume not mounted
36	24	ERREXTIO	Irrecoverable error on external lock file

Figure 47. Lock Manager Return Codes (LOCK Macro)

### Unlock Return Codes

Return Dec	Code Hex	Flag	Description
0	0		Request executed successfully
4	4		Resource is not locked for the issuing task/partition
8	8		DTL format error

Figure 48. Lock Manager Return Codes (UNLOCK Macro)



---

## Deadlock Detection

Assume that task T1 requests a resource, say RES1, which is already locked.

The owner chain of RES1 is scanned for owners who prevent T1 from locking this resource. If T1 itself is an owner of RES1 then a dead lock is detected.

The RESOURCE-BOUND owners (Task Status Byte, see Figure 26 on page 59) are entered into the dead lock test table (DLTT) and processed the same way as T1, owners that are not RESOURCE-BOUND are ignored.

This test is repeated for all entries of the DLTT (if there are any). Let's assume T2 is the first/next entry in the DLTT waiting for resource RES2. If T1 is an owner of RES2 then a dead lock is detected. The RESOURCE-BOUND owners are entered into the DLTT.

This testing is repeated until there are no more DLTT entries to be checked or until a dead lock is detected.

## Deadlock Test via Deadlock Test Table (DLTT)

The DLTT contains as many 2-byte entries as the maximum number of tasks specified for supervisor generation. If deadlock test is performed, the DLTT entries will contain the TIDs of the lock-bound (RURBND - x'8E') tasks. The pointer to the resource (LOCKTAB) on which a lock-bound task is waiting, will be found in the TIBSTATE. If the last bit of TIBSTATE is on, the task will lock the resource exclusively (E1 request).

### Notes:

E1 request: CONTROL=E, LOCKOPT=1

E2 request: CONTROL=E, LOCKOPT=2

## Internal Interface to Deadlock Test

If FETCH has to set a task into wait because the LTA is in use the deadlock test is called before. The deadlock test checks if a specific task (in this case LTA owner) is waiting for a resource which is owned by the actual running task (here the LTA requesting task).

A deadlock situation is indicated by setting the deadlock return code before leaving the deadlock test.

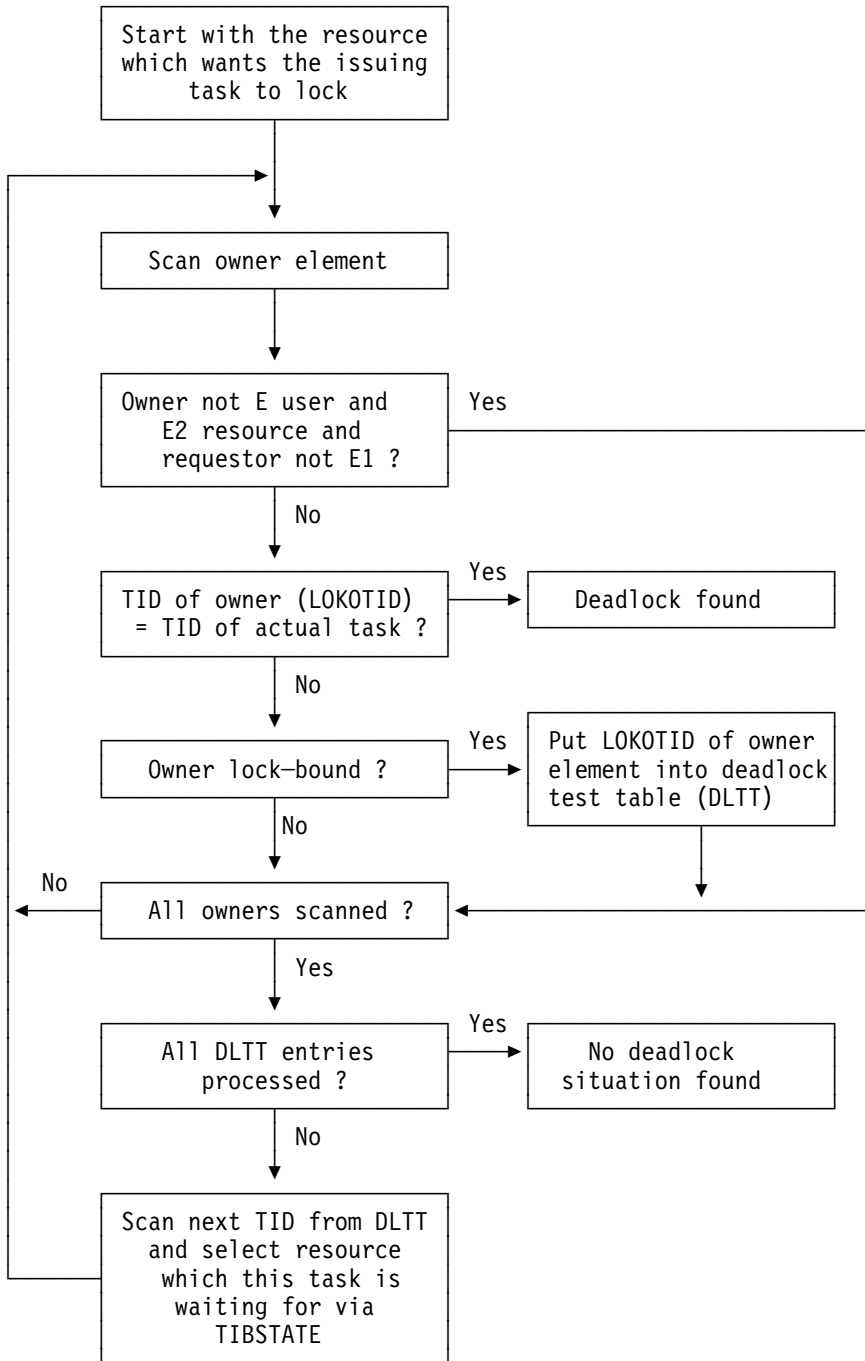


Figure 49. Deadlock Test

## Possible Deadlock Situations

1. External space is exhausted:
  - All resources of this block are owned by the issuing task or by a resource-bound task of this CPU (only deadlocks are detected which are caused by actions of one system).
2. Supervisor space is exhausted:
  - Waiting for a free owner element:
    - No owner element of this resource found, whose owning task is not resource bound (owner element has the TID of the requesting task).
  - Waiting for LOCKTAB space:
    - No LOCKTAB entry found, whose owners are all running (no owner is resource bound). Every LOCKTAB entry has just one owner element where its owning task is waiting for.
3. Resource is already locked:
  - Locked by the issuing task itself:
    - Deadlock if E1 request.
    - Deadlock if resource already locked with E1 by the issuing task.
    - Deadlock if resource already locked with E2 by the issuing task.
  - Not locked by the issuing task:
    - Find deadlock situation via deadlock test table (DLTT). (See paragraph: Deadlock Test)

## External Deadlock Checking

The described deadlock check routine can also be called by other supervisor routines. The entry point for external deadlock checking is SGLOCKCK. The calling function wants to set a task into wait state. By running the deadlock check routine it can be detected if this would lead to a deadlock situation with tasks waiting for locked resources.

The interface is described with:

INPUT R9	task id of deadlock candidate (actual task id)
INPUT R6	dispatcher address
INPUT R14	return address
OUTPUT R15	return code:
	0 = no deadlock
	16 = deadlock will occur
EXIT R6	in case of lockgate is not free
EXIT R14	normal return to caller

## DASD Sharing (Lock Manager)

When resources are locked across systems, the resource name and some control information are entered into the external lock file to assign the resource to this CPU.

When an externally locked resource is unlocked, the lock entry is removed from the external lock file, to allow other CPUs to lock the resource.

Within the SVC 110 (X'6E') processing routine it is not possible to issue SVC instructions. Therefore, the external lock file processing is done by a special system task, the Lock-System-Task (LCK). The LCK-Task is activated when the SVC X'6E' processing routine wants to read from or write to the external lock file. For additional information see description of "LOCK and UNLOCK (SVC 110 - X'6E')" on page 118.

## External Locking

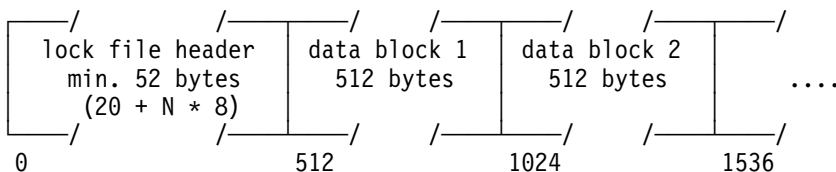
An external communication area, the external lock file, reflects at any time to all the sharing systems the system-wide locking status.

The external lock file is a system file which is shared among all sharing systems. Any resource to be locked across systems is contained in this external lock file.

The communication between the sharing systems is established during IPL via the DLF (Define Lock File) command. The VSE system which is IPLed first creates the external lock file. The other systems refer to this already created lock file, when they join the sharing environment.

## Lock File Format

The external lock file consists of a header block and data blocks. The header block contains a file description of the external lock file and information about the sharing CPUs. The data blocks contain the lock entries (resource name plus control information).



### Notes:

1. N = Number of CPUs
2. default 4 CPUs, max. 31 CPUs

Figure 50. Lock File Format

## Header Record Format

The lock file header record starts with a 20 byte file description of the lock file. The fields of this file description are identical with the first 20 bytes of the DLF Table in the supervisor.

This file description is followed by a list of the CPU IDs of the sharing systems. For any sharing CPU there is an 8-byte field containing two flag bytes and a 6-byte CPU identification.

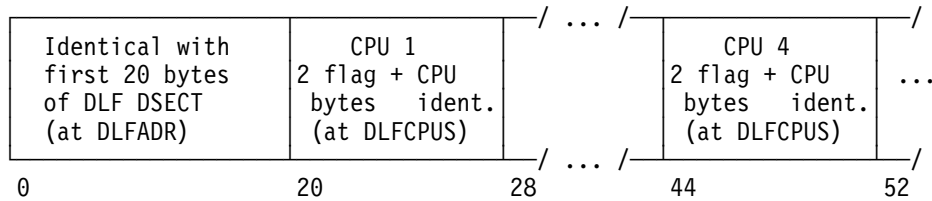


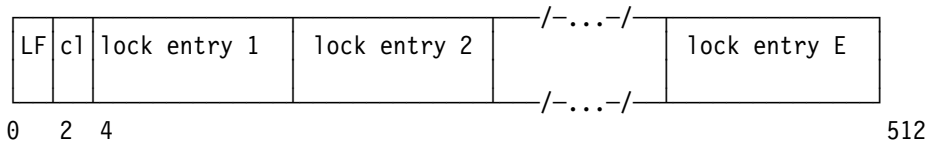
Figure 51. Lock File Header Format

### Lock File Data Blocks

The physical block length is 512 bytes for CKD devices. For FBA devices the physical block length equals the physical block length of the FBA device (presently always 512 bytes).

Each block contains a 2-byte identification field, a 2-byte count field and lock entries.

The identification field contains the characters 'LF' (Lock File) The count field contains the number of lock entries stored in this data block. The lock entries contain the 12-byte resource name and one lock byte for any sharing CPU (a minimum of 4 and a maximum of 31 bytes).



c1 = Count of lock entries in this data block  
 E = Maximum possible number of lock entries

Figure 52. Lock File Data Block Format

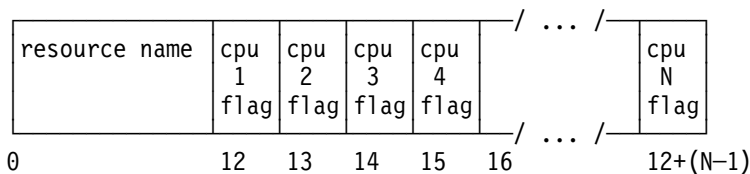


Figure 53. Lock File Entry Format

Flag	Appr.	Description
x'00'		no locking
x'01'	S1	CONTROL=S LOCKOPT=1
x'11'	E1	CONTROL=E LOCKOPT=1
x'02'	S2	CONTROL=S LOCKOPT=2
x'12'	E2	CONTROL=E LOCKOPT=2
x'04'	S4	CONTROL=S LOCKOPT=4
x'14'	E4	CONTROL=E LOCKOPT=4

Figure 54. CPU N Flag

### Lock File Block Capacity

The length of one lock entry depends on the number of sharing CPUs. The maximum number of lock entries which may be stored into one disk block is dependent on the number of sharing CPUs (max. 31) and on the data block length (presently always 512 bytes).

The number of sharing CPUs is restricted to 31.

Example:	
Number of sharing CPU:	4
Length of one lock entry (resource name length + no. of CPUs):	16
Length of available space in one data block (512 - (2 byte ID + 2 byte count)):	508
-----	
Number of lock entries per data block (length of avail. space DIV length of lock entry):	31

Figure 55. Maximum Number of Lock Entries in One Data Block (ex. 4 CPUs)

## Mapping of Locks into Disk Blocks

Locked resources are stored into the external lock file at random. A hashing algorithm maps the resource name into the disk block number. This is done to spread the lock entries evenly over the external lock file. Within the disk block, lock entries are stored on the next free place.

When a lock entry is deleted, the last lock entry is moved to the free place.

### Hashing Algorithm

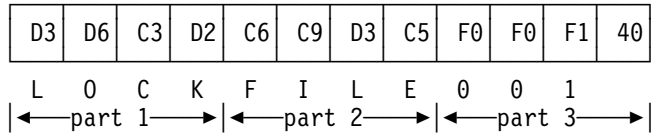
1. Compress the 12-byte resource name by two EXCLUSIVE OR instructions into a full word.
2. Divide this full word by the number of blocks in the lock file.
3. You will get the relative block number within the external lock file, if you use the remainder of this division and add one block (for the header record block).

### Example

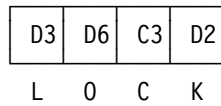
Ex.: Look for resource "LOCKFILE001" and compute disk block number.

Number of blocks in our example: X'25' (=DLFNBLK)

Resource name (12 bytes):

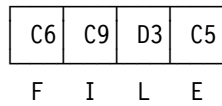


Part 1:



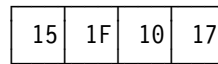
XOR

Part 2:

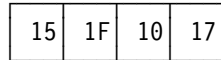


=

Result 1:

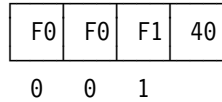


Result 1:



XOR

Part 3:

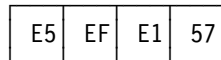


=

Result 2:

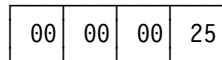


Result 2:



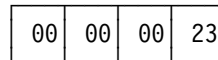
MOD

Number of blocks:



=

Remainder:



1 header record block	+	1	
Disk block number of lock entry (=DLFHBLK)		24	

Figure 56. Mapping of Locks into Disk Blocks

### Lock File Size

During IPL the lock file size is determined.

### Lock Entry - Storing and Retrieval

*Record insertion:* New lock entries are entered into the first free place of the selected block (selected via hashing). Records within one block are not ordered.



---

3	REC1	REC2	REC3	RECx	RECy	...	before
---	------	------	------	------	------	-----	--------

Number  
of rec.

4	REC1	REC2	REC3	REC4		...	after
---	------	------	------	------	--	-----	-------

Number  
of rec.

---

*Record retrieval:* Scan the whole block to find the required lock entry.

*Record deletion:* When a lock entry is deleted, the last lock entry is moved to the free place (to keep the block 'dense').

Example: REC2 is deleted, REC4 is moved to the free place.

---

4	REC1	REC2	REC3	REC4	RECx	...	before
---	------	------	------	------	------	-----	--------

Number  
of rec.

3	REC1	REC4	REC3	REC4	RECx	...	after
---	------	------	------	------	------	-----	-------

Number  
of rec.

---

### Fetch in a DASD Sharing Environment

For FETCH (Program Retrieval) in a DASD Sharing Environment see "DASD Sharing Environment" on page 360.

---

## Service and Debugging Information

The here described data areas can be changed at any time by IBM if necessary and cannot be considered as an interface. Any interpretation, dependency or conclusion is only valid for the shown environment.

### Overview

#### Basics

<b>LOCK</b>	Protects resources against simultaneous access by other tasks.
<b>UNLOCK</b>	Frees locked resources.
<b>VSE SVC</b>	LOCK/UNLOCK is SVC 110 (X'6E).
<b>USE/RELEASE</b>	(SVC 63/64) are still supported without interface macro.
<b>LOCK Gate</b>	The SVCs are serialized with system gate SRQUSE (X'8B).

#### Resources

- Libraries
- Catalogs
- Data sets
- Data (e.g. control blocks)

#### Protection Scope

<b>Internal Locking</b>	Protects resources against other tasks or partitions.
<b>External Locking (DASD Sharing)</b>	Protects resources in the present system and all sharing systems.

#### Lock Manager Macros

<b>Define the Lock (DTL)</b>	Parameter List for LOCK/UNLOCK, generated at compile time
<b>GENDTL</b>	DTL, generated at run time
<b>MODDTL</b>	Modifies the DTL contents
<b>LOCK</b>	Own a resource
<b>UNLOCK (ALL)</b>	Free a resource or all held resources by this task/partition

## DTL Mapping

MAPDTL	DSECT		DSECT FOR DEFINE THE LOCK
DTLLENG	DS	AL2 (DTLLEN)	LENGTH OF DTL
DTLFLG1	DS	XL1 '10'	CONTROL AND LOCKOPT SPECIFICATION
DTLEXC	EQU	X'10'	CONTROL=E
DTLOPT1	EQU	1	LOCKOPT=1
DTLOPT2	EQU	2	LOCKOPT=2
DTLOPT4	EQU	4	LOCKOPT=4
DTLFLG2	DC	XL1 '00'	JC & VSAM FLAGS
DTLKEEP	EQU	X'80'	KEEP UNTIL EOJ
DTLPART	EQU	X'40'	OWNER=PARTITION
DTLREDC	EQU	X'20'	REDUCE STRENGTH OF LOCK (UNLOCK ONLY)
DTLEXTR	EQU	X'10'	SCOPE=EXTERNAL
DTLVOL	EQU	8	VOLID SPECIFIED
DTLNAME	DS	CL12 'DUMMYMAPDTL '	RESOURCE NAME
DTLVOLID	DS	CL6 ' '	VOLUME IDENTIFICATION
DTLECB	DS	XL1 '0'	ECB OF REQUESTING TASK
DTLRC	DS	XL1 '0'	RETURN CODE OF THE REQUEST
DTLPOST	DS	XL1 '0'	FLAG TO POST THE TASK
DTLECB3	DS	XL1 '0'	BYTE 3 OF ECB
	DS	XL4 '0'	RESERVED
DTLLEN	EQU	*-MAPDTL	LENGTH OF DTL

## LOCK Function

Request the usage of a resource. Beside the DTL a FAIL parameter defines the system action in case of unsuccessful completion.

**FAIL=RETURN** The requesting task always gets control back and has to check the return code for the SVC's response.

**FAIL=WAITC** The system places the task in a wait state (state X'8E, waiting for locked resource), if the resource cannot be obtained currently.

all other cases the task gets control back and has to check the return code.

case a resource cannot be obtained because it is in use by another system, the lock system task retries the lock request every second (there is no other way to recognize that the resource becomes free).

**FAIL=WAIT** The requesting task gets control back, when it's owning the resource.

If the resource is locked by another task or if lock file or DLF space is exhausted then the task is set into a wait state (X'8E).

In case a resource cannot be obtained because it is in use by another system, the lock system task retries the lock request every second.

In all other unsuccessful cases the task is cancelled.

**FAIL=WAITECB** The requesting task always gets control back and has to check the return code.

If the resource is currently locked by another task, then the request is queued to the resource.

As soon as the resource is UNLOCKed the lock system task repeats this lock request and tries to lock it for this task.

In case a resource cannot be obtained because it is in use by another system, the lock system task retries the lock request every second.

If the task becomes owner of the resource the ECB in the DTL is posted to indicate completion of the former lock request.

## UNLOCK Function

Free a resource. All tasks waiting for this resource are posted (FAIL=WAIT/WAITC) and retries for existing requests from tasks with FAIL=WAITECB are performed.

IF parameter 'ALL' is specified, all resources held by that task are released (except KEEP=YES). In this case FREEVIS of the no longer used control blocks is performed.

UNLOCK ALL does not provide a return code.

## Data Lock File (DLF)

- The lock file resides on a DASD which must be ADDED with the option SHR.
- The lock file contains all external locks from all sharing systems.
- The access to the lock file is established with the IPL DLF statement. Either CUU or VOLSER must be specified and further parameters specify extent and size of the lock file and the number of sharing systems.
- Lock file I/O is protected via channel commands reserve/release (X'B4/94). Therefore it's not recommended to place performance critical data on that CUU.
- The lock file format is the same for all VSE releases.
- The lock file can only be formatted while all sharing systems are down.
- Formatting of the lock file can be done by any sharing system.
- Formatting of the lock file is necessary if
  - A new storage medium is used
  - The number of sharing systems is increased

## Sharing Systems

- Each sharing system needs a unique CPU ID.
- If all VSE systems are guests of the same VM system, then the lock file can be defined on a VM virtual disk.
- In all other cases, like VSE native, VSE native in LPAR, VSE guests in more than one VM or a combination of the above, the lock file has to reside on a FBA, CKD or ECKD DASD with sharing capability.

## UNLOCK SYSTEM

- If one of the sharing systems ends up in hardwait or softwait it may block all other systems because of held locks.
- The AR command UNLOCK SYSTEM=cpuid performs an UNLOCK ALL on the lock file for the specified system. The header entry for this system is also cleared.

## Data Structures

The following examples are taken from dumped storage.

### Lock File Header Format

The lock file header's length is  $20 + ncpu * 8$  bytes.

Example for a lock file header:

```
D3C60004 020002B1 001F0010 0001002E 06 *LF      £      *
000F030C 8000FFFF 10009221 00000000 06 *    0      k      *
00000000 00000000 00000000 00000000 06 *
00000000 00000000 00000000 00000000 06 *
```

block indicator

```

| | | | |
| | | | | number of sharing systems (cpu fields)
| | | | |
| | | | | physical block length
| | | | |
| | | | | number of data blocks in lock file
| | | | |
| | | | | number of entries per block
| | | | |
| | | | | length of one entry (12+ncpu)
| | | | |
| | | | | cylinder address of lock file
| | | | |
| | | | | number of blocks per track
| | | | |
vvvvvvvv vvvvvvvv vvvvvvvv vvvvvvvv
D3C60004 020002B1 001F0010 0001002E 06 *LF      £      *
```

number of tracks per cylinder

```

| | | | |
| | | | | device type = rps ckd (fba=1, ckd=2, eckd=4)
| | | | |
| | | | | device type code = ckd dasd
| | | | |
| | | | | flag1 80 = this cpu field is in use, else 00
| | | | |
| | | | | flag2 not used
| | | | |
| | | | | cpu id (sharing system identification)
| | | | |
| | | | | currently only 1 system
vvvvvvvv vvvvvvvv vvvvvvvv
000F030C 8000FFFF 10009221 00000000 06 *    0      k      *
00000000 00000000 00000000 00000000 06 *
00000000 00000000 00000000 00000000 06 *
```

## Lock File Data Format

A data block is always 512 bytes long

Example for a lock file data block:

```
D3C60001 D9C5E2D6 E4D9C3C5 60C5F1F7 06 *LF RESOURCE-E17*
11000000 00000000 00000000 00000000 06 *
00000000 00000000 00000000 00000000 06 *
```

block indicator

```

|||
||| number of entries in this block
|||
||| resource name
|||
vvvvvvvv vvvvvvvv vvvvvvvv vvvvvvvv
D3C60001 D9C5E2D6 E4D9C3C5 60C5F1F7 06 *LF RESOURCE-E17*
```

```

in use by slot 0, exclusive = 1x option 1 = y1
|| shared = 0x option 2 = y2
|| option 4 = y4
|| not used = 00
```

other slots

```

|||
||| not yet used
vvvvvvvv
11000000 00000000 00000000 00000000 06 *
00000000 00000000 00000000 00000000 06 *
```

## Lock Table Format

- The Lock table contains all locks of one VSE system.
- All entries are chained in a double linked list, the anchor point to the table is at label ALOKTABA in the supervisor. ALOKTABA is the first fullword after eye catcher ILCKSP.
- The control block, containing the resource name and the locking status is called LOCKTAB ENTRY. Pointers refer to
  - OWNER ELEMENTS (owner's task ID) and
  - REQUEST ELEMENTS (tasks which previously issued LOCK with FAIL=WAITECB and resource was in use).
- A locktab entry is 32 bytes long, owner and request element's length is 16 bytes.



Example for a locktab entry with owner and requestors:

```
V0004F9D4 00311080 003110C0 D9C5E2D6 E4D9C3C5 * 0 {RESOURCE*
V0004F9E4 60C5F1F0 11800001 0004F9F4 00000000 *-E10 0 94 *
```

```
V00311080 00000000 00250000 00011000 00000000 * *
```

```
V003110C0 0004FA74 00240000 00000000 00600ABE * 8E - '*
V0004FA74 00000000 00230000 00000000 00600ABE * - '*
```

LOCKTAB ENTRY:

```

      pointer to first owner element
      | | | | |
      | | | | | pointer to first request element
      | | | | |
      | | | | | resource name
      | | | | | | | | | | | | | | | |
      vvvvvvvv vvvvvvvv vvvvvvvv vvvvvvvv
V0004F9D4 00311080 003110C0 D9C5E2D6 E4D9C3C5 * 0 {RESOURCE*
```

resource name (contd.)

```

      | | | | |
      | | | | | locking status: exclusive = 1x option 1 = y1
      | | | | | shared = 0x option 2 = y2
      | | | | | option 4 = y4
      | | | | | not used = 00
      | | | | |
      | | | | | flag: 80 = entry in use, 40 = owner=partition
      | | | | | 20 = s user waits, 10 = entry in lock file
      | | | | | 08 = e user waits
      | | | | |
      | | | | | total number of exclusive users
      | | | | |
      | | | | | pointer to next entry
      | | | | |
      | | | | | pointer to previous entry
      | | | | | | | | | | | | | | | |
      vvvvvvvv vvvvvvvv vvvvvvvv vvvvvvvv
V0004F9E4 60C5F1F0 11800001 0004F9F4 00000000 *-E10 0 94 *
```

OWNER ELEMENT:

```

pointer to next owner element
| | | | |
| | | | | owner's task id
| | | | |
| | | | | number of shared users
| | | | |
| | | | | number of exclusive users
| | | | |
| | | | | flag: 80 = keep until end of job
| | | | | 10 = exclusive usage
| | | | |
| | | | | reserved
vvvvvvvv vvvvvvvv vvvvvv
V00311080 00000000 00250000 00011000 00000000 * *
```

REQUEST ELEMENT 1:

```

pointer to next request element
| | | | |
| | | | | requestor's task id
| | | | |
| | | | | user's dtl address
| | | | |
| | | | | reserved
vvvvvvvv vvvv vvvvvvvv
V003110C0 0004FA74 00240000 00000000 00600ABE * ¼È - ”*
```

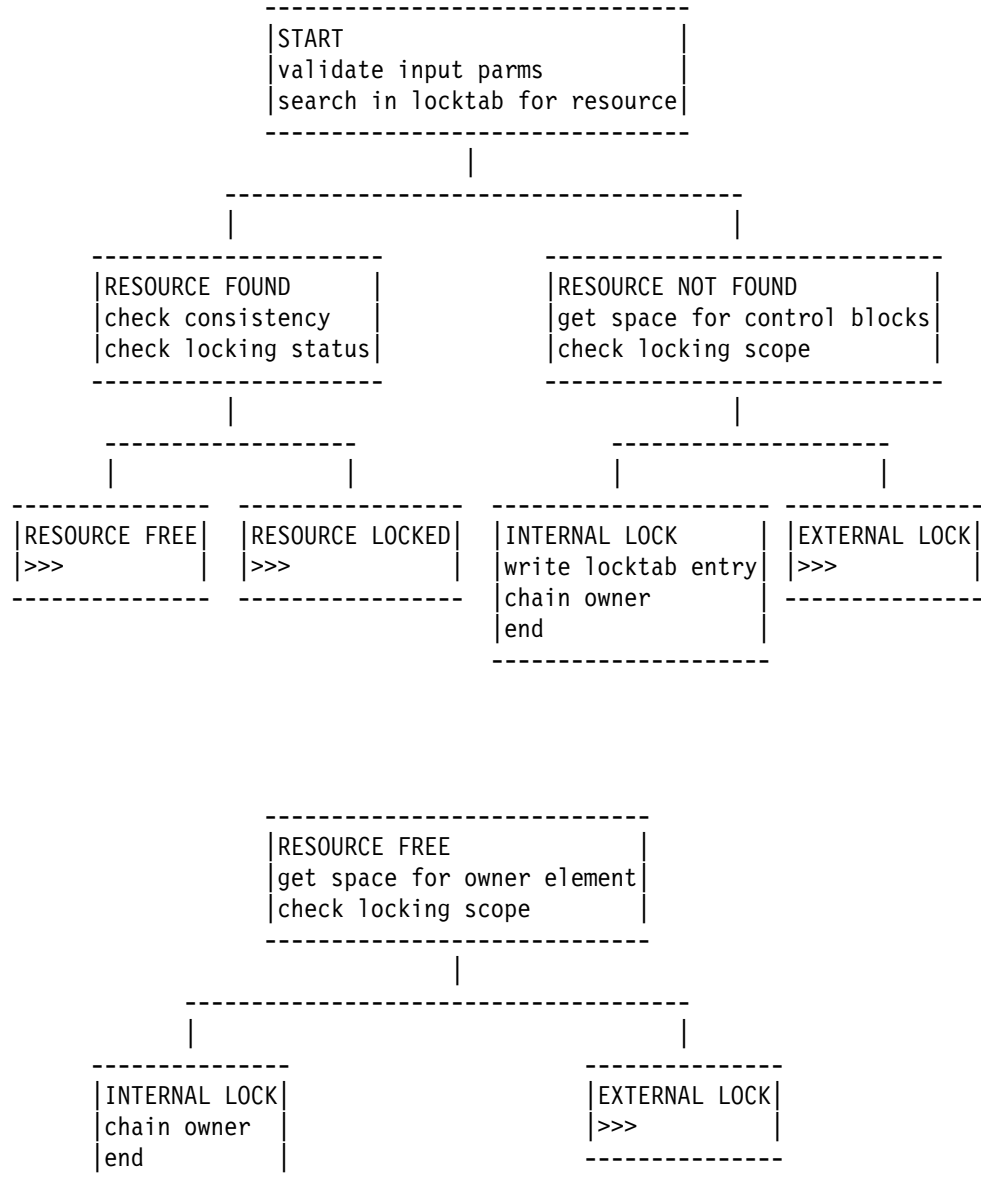
REQUEST ELEMENT 2:

```

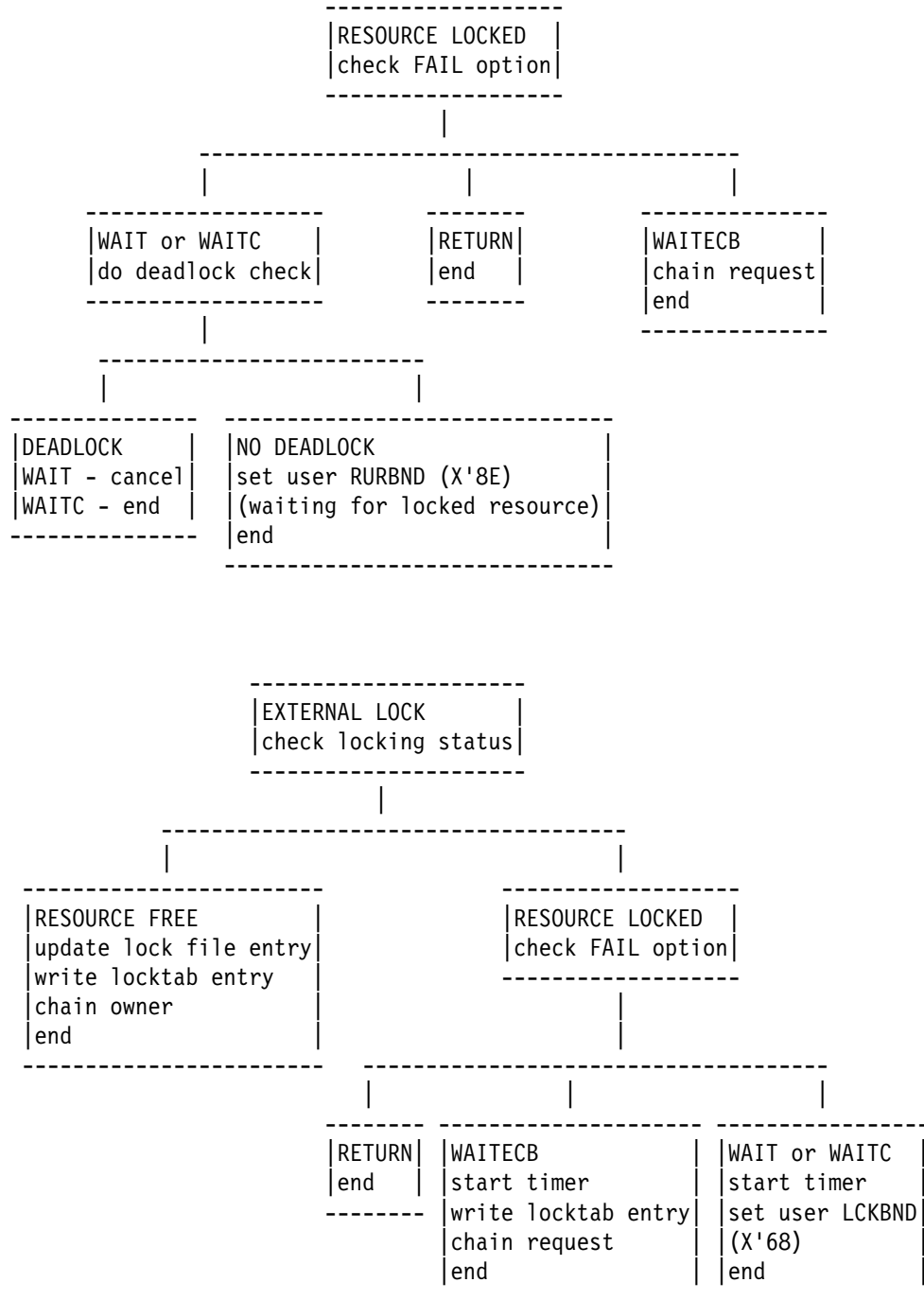
V0004FA74 00000000 00230000 00000000 00600ABE * - ”*
```

# Algorithms

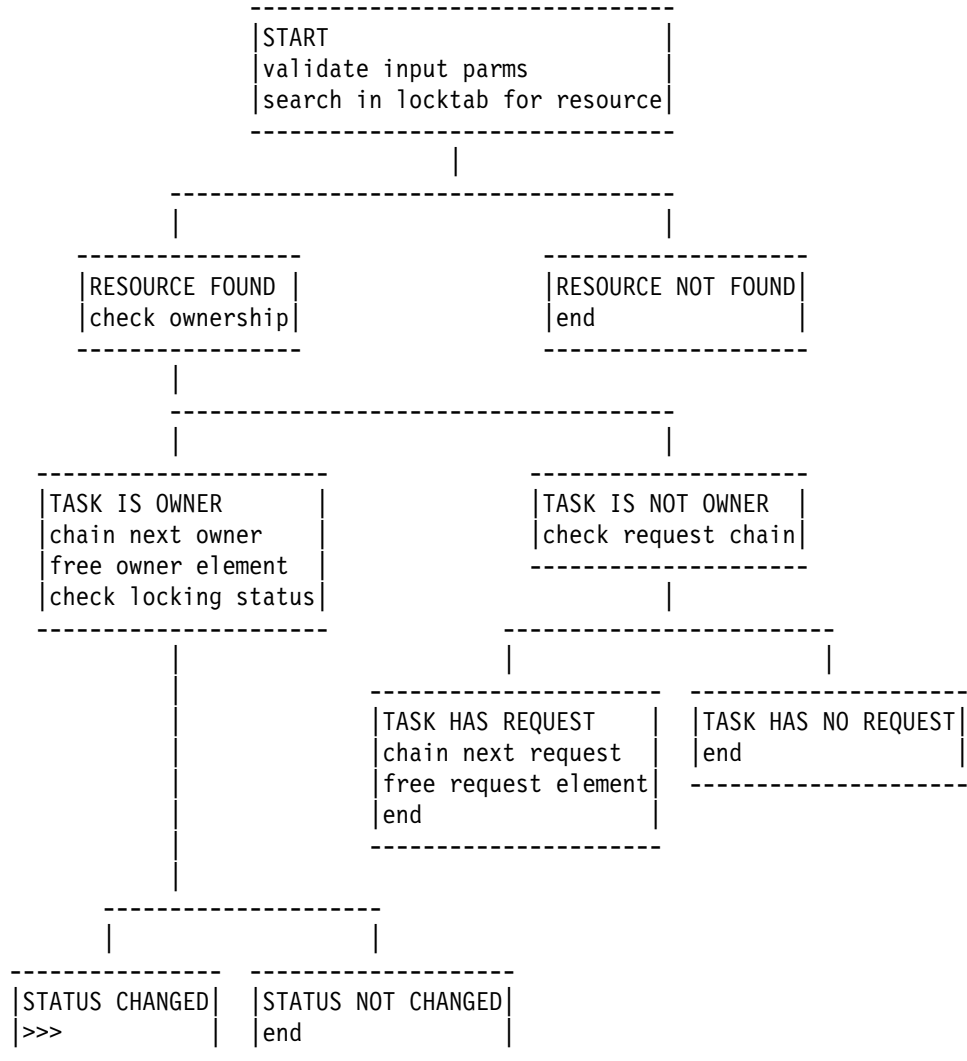
## Lock Algorithm



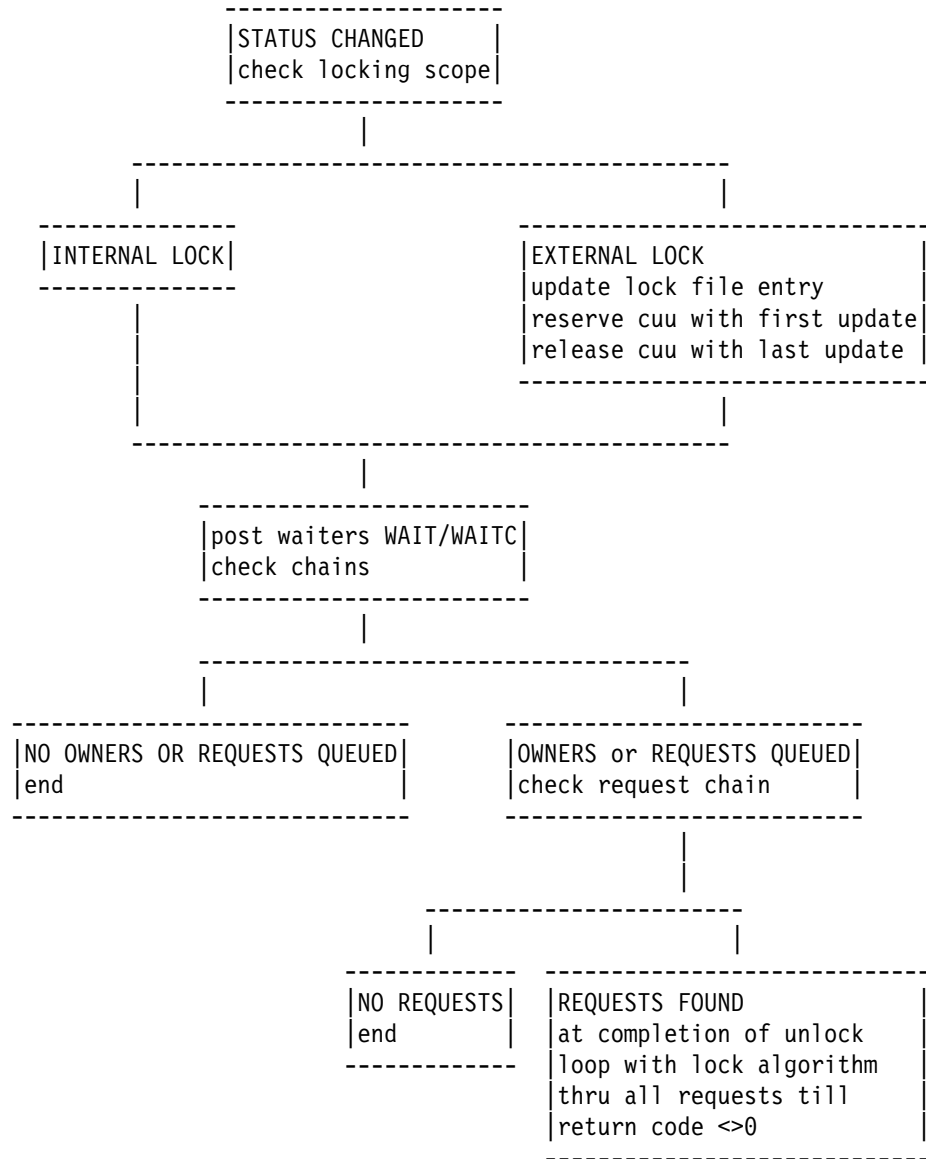
## Lock Algorithm cont...



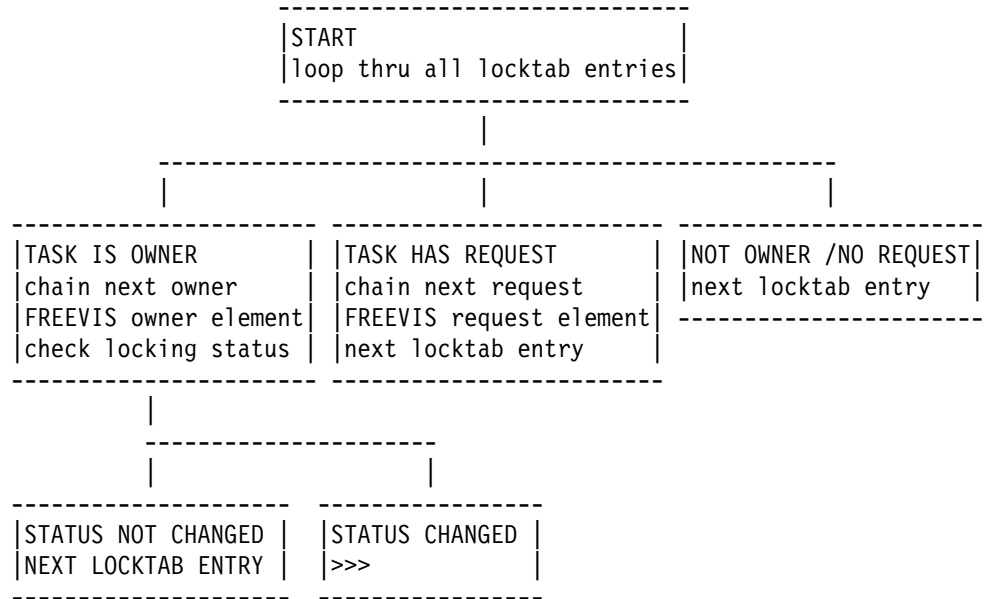
## Unlock Algorithm



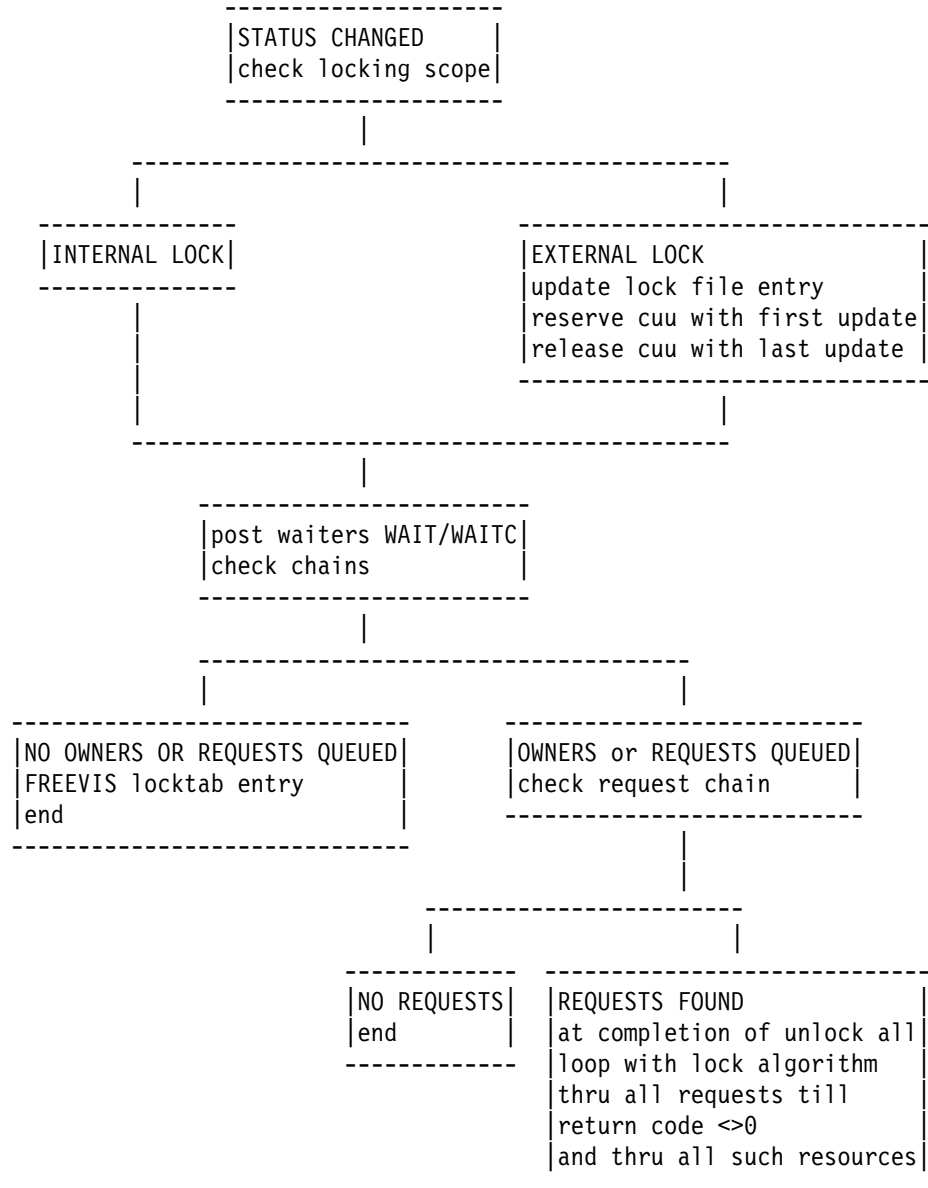
## Unlock Algorithm cont...



## Unlock All Algorithm



## Unlock All Algorithm cont...



## IPL Algorithm





## Lock Manager Trace Area

Eye catcher is LCKT in SGLOCK. A trace entry is 64 bytes long. The first 3 fullwords specify start, end and actual position in the wrap around area.

Example for trace entries:

```
V0004FAD0 ..... D3C3D2E3 0004FAE8 06 *.....LCKT ½Y*
V0004FAE0 000504E7 0004FB68 24240700 006092C4 06 * X Üy -kD*
V0004FAF0 11000000 01000000 5BD1D6C2 C1C3C3E3 06 * $JOBACCT*
V0004FB00 40404040 0004FA34 0004FA74 00000000 06 * ½ ½È *
V0004FB10 5BD1D6C2 C1C3C3E3 40404040 11800001 06 * $JOBACCT Ø *
V0004FB20 0004FA14 00000000 24240100 006092C4 06 * ½ -kD*
V0004FB30 11000024 01000F00 5BD1D6C2 C1C3C3E3 06 * $JOBACCT*
V0004FB40 40404040 0004FA34 00000000 00000000 06 * ½ *
V0004FB50 00000000 00000000 00000000 00000000 06 * *
V0004FB60 0004FA14 00000000 ..... 06 * *
```

```

                                pointer to first byte of area
                                |||||
                                vvvvvvvv
V0004FAD0 ..... D3C3D2E3 0004FAE8 06 *.....LCKT ½Y*
```

```

pointer to last byte of area
|||||
pointer to next free entry in area
|||||
LAST BUT ONE ENTRY:
task id as used for owner
|||||
actual running task (low core)
|||||
lock/unlock req.: 00 release
                  01 unlock
                  03 lock fail=return
                  06 use
                  07 lock fail=wait
                  0B lock fail=waitc
                  21 unlock all
                  31 unlock all eoj
                  41 unlock sysid
                  43 deadlock check
                  83 lock fail=waitecb
return code (see return codes)
|||||
user's dtl address
|||||
vvvvvvvv vvvvvvvv vvvvvvvv vvvvvvvv
V0004FAE0 000504E7 0004FB68 24240700 006092C4 06 * X Üy -kD*
```



## Statistic Counters

At eye catcher COUNTERS in SGLOCK there are fields for LOCK/UNLOCK request return code tracking:

- Internal requests
- External requests
- Internal LOCKs
- External LOCKs
- UNLOCKs
- deadlock checks
- Lock file gets
- Lock file puts

```
LOCKCNTS DC      CL8 'COUNTERS'  
SVC110IN DC     A(0)    INTERNAL LOCKING COUNTER  
SVC110EX DC     A(0)    EXTERNAL LOCKING COUNTER  
LOCKRC00 DC     A(0)    LOCK COUNTER FOR RC = 0  
ELCKRC00 DC     A(0)    LOCK COUNTER FOR RC = 0  
LOCKRC04 DC     A(0)    LOCK COUNTER FOR RC = 4  
ELCKRC04 DC     A(0)    LOCK COUNTER FOR RC = 4  
LOCKRC08 DC     A(0)    LOCK COUNTER FOR RC = 8  
ELCKRC08 DC     A(0)    LOCK COUNTER FOR RC = 8  
LOCKRC0C DC     A(0)    LOCK COUNTER FOR RC = 12  
ELCKRC0C DC     A(0)    LOCK COUNTER FOR RC = 12  
LOCKRC10 DC     A(0)    LOCK COUNTER FOR RC = 16  
ELCKRC10 DC     A(0)    LOCK COUNTER FOR RC = 16  
LOCKRC14 DC     A(0)    LOCK COUNTER FOR RC = 20  
ELCKRC14 DC     A(0)    LOCK COUNTER FOR RC = 20  
LOCKRC18 DC     A(0)    LOCK COUNTER FOR RC = 24  
ELCKRC18 DC     A(0)    LOCK COUNTER FOR RC = 24  
LOCKRC1C DC     A(0)    LOCK COUNTER FOR RC = 28  
ELCKRC1C DC     A(0)    LOCK COUNTER FOR RC = 28  
LOCKRC20 DC     A(0)    LOCK COUNTER FOR RC = 32  
ELCKRC20 DC     A(0)    LOCK COUNTER FOR RC = 32  
LOCKRC24 DC     A(0)    LOCK COUNTER FOR RC = 36  
ELCKRC24 DC     A(0)    LOCK COUNTER FOR RC = 36  
ULOCKRC0 DC     A(0)    UNLOCK COUNTER FOR RC = 0  
ULOCKRC4 DC     A(0)    UNLOCK COUNTER FOR RC = 4  
ULOCKRC8 DC     A(0)    UNLOCK COUNTER FOR RC = 8  
ULOCKRCC DC     A(0)    UNLOCK COUNTER FOR RC = 12  
DEADRC00 DC     A(0)    DEADLOCKCHECK WITH RC = 0  
DEADRC04 DC     A(0)    DEADLOCKCHECK WITH RC = 4  
IOGET   DC      A(0)    READ I/O'S TO LOCKFILE  
IOPUT   DC      A(0)    WRITE I/O'S TO LOCKFILE
```

## Error on Lock File

Message **0T01E ERROR ON LOCK FILE** indicates one of these situations:

1. user error in DLF statement
2. an unrecoverable I/O error
3. lock file format error
4. lock file logical error

The investigation of type 1, 2 and 3 errors does not necessarily require a stand-alone dump. Type 4, which is the most frequent one, however needs analysis of the current lock manager data at the point of failure. The system does not stop processing after issuing the message and therefore a dump at a later point might not show the reason of the problem. In order to speed up the error analysis of such situations a method to circumvent the standard procedure

- take standalone dump
- mail to IBM
- analyze a dump

has to be established.

The lock manager writes together with the message **0T01E ERROR ON LOCK FILE** additional data that describes the error situation. This information can be sent in for analysis via FAX.

## Example

this example shows an inconsistency between lock table and lock file when an UNLOCK function was performed.

```

F4 0025 0T01E ERROR ON LOCK FILE
F4 0025 LOCK MANAGER EMERGENCY DATA
V00051138 000C0025 0101000F 80000000 0025D9C5 *      Ø      RE*  R00051138
V00051148 E2D6E4D9 C3C560C5 F1F01110 00051148 *SOURCE-E10  ç*  R00051148
F4 0025 LOCKTAB ENTRY
V0004FE70 00000000 00000000 00000000 00000000 *          *  R0004FE70
V0004FE80 00000000 00000000 0004FE90 00000000 *          Ū°  *  R0004FE80
F4 0025 LOCK FILE DISK BLOCK
V00086198 D3C60000 D9C5E2D6 E4D9C3C5 60C5F1F0 *LF  RESOURCE-E10*  R00086198
V000861A8 11000000 00000000 00000000 00000000 *          *  R000861A8
V000861B8 00000000 00000000 00000000 00000000 *          *  R000861B8
V000861C8 00000000 00000000 00000000 00000000 *          *  R000861C8
. . . . .
F4 0025
V00086358 00000000 00000000 00000000 00000000 *          *  R00086358
V00086368 00000000 00000000 00000000 00000000 *          *  R00086368
V00086378 00000000 00000000 00000000 00000000 *          *  R00086378
V00086388 00000000 00000000 00000000 00000000 *          *  R00086388
F4 0025 LOCK MANAGER TRACE AREA
V0004FF78 D3C3D2E3 0004FF88 00050987 00050788 *LCKT  h  g  h*  R0004FF78
F4 0025 LOCK MANAGER TRACE AREA
V0004FF88 3F3F0B00 00902D80 01080000 01000000 *      ° Ø      *  R0004FF88
V0004FF98 E5C3E3E2 F2F2F0E4 D7D30000 0033B160 *VCTS220UPL  £-*  R0004FF98
V0004FFA8 0033B140 00000000 E5C3E3E2 F2F2F0E4 *  £      VCTS220U*  R0004FFA8
V0004FFB8 D7D30000 01800000 0033B180 0033B120 *PL  Ø  £Ø  £ *  R0004FFB8
. . . . .
F4 0025
V00050748 25250B00 00600B90 01100000 01000000 *      - °      *  R00050748
V00050758 D9C5E2D6 E4D9C3C5 60C5F1F7 0033B520 *RESOURCE-E17 § *  R00050758
V00050768 0033B510 00000000 D9C5E2D6 E4D9C3C5 *  §      RESOURCE*  R00050768
V00050778 60C5F1F7 01900000 00000000 0033B4E0 *-E17 °      fl\ *  R00050778
. . . . .
F4 0025
V00050948 3F3F0104 00902D80 11000000 01000000 *      ° Ø      *  R00050948
V00050958 E5F24000 9042CCC3 C1E70000 00000000 *V2  °äöCAX      *  R00050958
V00050968 000A0000 00000000 00000000 00000000 *          *  R00050968
V00050978 00007480 00003F00 070C0000 8007146A *  ÈØ      Ø  ]*  R00050978

```

Lock file emergency data contains:

- 2 bytes: TID (low core)
- 2 bytes: LCKUTID (lock manager TID, for which the service runs)
- 1 byte : LOCKPARM (already explained)
- 1 byte : LOKOWORK (work area to compare DTL info)
- 1 byte : REQWORK (work flag for deadlock test)
- 1 byte : UNLCKFLG (flag for unlock service)
- 1 byte : UNLCKFLG (flag for unlock service:
  - 01 activate waiting tasks
  - 02 free owner element
  - 04 free locktab entry
  - 08 activate E1 requestors
  - 10 lock file block modified
- 1 byte : DSHRFLG (flag for system task:
  - 80 system task is active
  - 40 timer request already set
  - 20 update on lock file required
  - 10 disk drive reserved
- 2 bytes: LCKCNT (count locked tasks, deadlock check)
- 2 bytes: TRCOTID (first owner's TID, if the resource is in use)
- 12bytes: TRCRESN (current resource name)
- 1 byte : TRCFLG1 (user's DTL option byte 1)
- 1 byte : TRCFLG2 (user's DTL option byte 2)
- 4 byte : reserved (ignore contents)

### Analysis of the Example

The UNLOCK request came for resource RESOURCE-E10 from task 25(F4). Current task is lock task C.

The UNLOCK was performed successfully in the lock table since the locktab entry was cleared.

However this resource was an external one. TRCFLG2 (DTLFLG2) shows X'10', scope=external.

The corresponding lock file block shows 0 as number of entries. It still can be seen that RESOURCE-E10 has been placed in this block before.

The lock manager detects the mismatch between local lock table and lock file and issues the error message.

The cause of such a type of error can be

- There are several VSE systems under one VM and 2 of them are running with the same CPUID.
- The reserve/release mechanism did not work. Software caching did suppress the lock manager's CCW's but did not provide a proper exclusive access handling. Software caching is not done in VSE. VM or vendor products (on VM or VSE) are candidates.

## Display Facility

To allow faster diagnosis in case of resources being in use or deadlock situations it is possible to display the actual locking status of

- the entire lock table
- all locks held by a specified partition (PIK)
- a given resource name
- a set of resource names, specified with name\*
- a set of resource names held by a given partition

This feature is available via Attention Routine command **LOCK SHOW**. Examples for the usage of the command are

<b>LOCK SHOW</b>	shows <b>all</b> currently held locks (resources) of this <b>VSE system</b> .
<b>LOCK SHOW=F4</b>	shows <b>all</b> currently held locks (resources) of <b>partition F4</b> .
<b>LOCK SHOW,RESOURCE-E10</b>	shows <b>who</b> is currently holding resource <b>RESOURCE-E10</b> (if held at all).
<b>LOCK SHOW,VSYSOPEN'00000001</b>	shows <b>who</b> is currently holding resource <b>VSYSOPEN00000001</b> (if held at all), where VSYSOPEN is interpreted as characters and 00000001 as hex value.
<b>LOCK SHOW,RESOURC*</b>	shows <b>who</b> is currently holding resources, starting with the character string <b>RESOURC</b> .
<b>LOCK SHOW=F6,RES*</b>	shows, if <b>partition F6</b> is currently holding resources, starting with the character string <b>RES</b> .

For interpretation of the output please refer to the lock table description.



## Example

Lock show

```
AR 0025 LOCKTAB ENTRY
V0004F344 003245A0 00000000 D9C5E2D6 E4D9C3C5 * äff RESOURCE* R0004F344
V0004F354 60C5F1F0 11900001 0004F364 00000000 *-E10 ° 3Ã * R0004F354
AR 0025 OWNER ELEMENT
V003245A0 00000000 002C0000 00011000 00000000 * * R0087F5A0
AR 0025 LOCKTAB ENTRY
V0004F364 0004F3E4 00000000 C4E3E2E5 C5C3E3C2 * 3U DTSVECTB* R0004F364
V0004F374 40404040 11800001 0004F384 0004F344 * Ø 3d 3ã* R0004F374
AR 0025 OWNER ELEMENT
V0004F3E4 00000000 00400000 00011000 00000000 * * R0004F3E4
AR 0025 LOCKTAB ENTRY
V0004F384 0004F3F4 00000000 E5C3E3E2 F2F2F000 * 34 VCTS220 * R0004F384
V0004F394 00000000 04C00000 0004F3A4 0004F364 * { 3u 3Ã* R0004F394
AR 0025 OWNER ELEMENT
V0004F3F4 00000000 00230001 00000000 00000000 * * R0004F3F4
AR 0025 LOCKTAB ENTRY
V0004F3A4 0004F414 00000000 E5C3E3E2 F2F2F000 * 4 VCTS220 * R0004F3A4
V0004F3B4 00000001 04C00000 00324020 0004F384 * { 3d* R0004F3B4
AR 0025 OWNER ELEMENT
V0004F414 00000000 00230001 00000000 00000000 * * R0004F414
. . . . .
AR 0025 LOCKTAB ENTRY
V00324440 003245E0 00000000 D9C5E2D6 E4D9C3C5 * ä\ RESOURCE* R0087F440
V00324450 60C5F1F7 01900000 00000000 00324420 *-E17 ° ä * R0087F450
AR 0025 OWNER ELEMENT
V003245E0 00000000 002C0001 00000000 00000000 * * R0087F5E0
AR 0015 1I40I READY
```

## Trace Facility

A trace facility for unsuccessful locks (RC<>0) and unlocks is provided. Trace contents can be specified according to the display facility from above. Besides that a command for trace deactivation is available. Traces can be done for

- all tasks and resources
- all resources belonging to a specified partition (PIK)
- a set of resource names, starting with name\* for all tasks
- a set of resource names and a specified partition

The Attention Routine command **LOCK TRACE** controls the various trace options. Examples for the usage of the command are

<b>LOCK TRACE</b>	traces <b>all</b> unsuccessful locks and unlocks of <b>all partitions</b> .
<b>LOCK TRACE=OFF</b>	sets the trace facility <b>off</b> .
<b>LOCK TRACE=F4</b>	traces <b>all</b> unsuccessful locks and unlocks of <b>partition F4</b> .
<b>LOCK TRACE,RESOURCE-E10</b>	traces <b>all</b> unsuccessful locks and unlocks of <b>RESOURCE-E10</b> of <b>all partitions</b> .
<b>LOCK TRACE,VSYSOPEN'00000001</b>	traces <b>all</b> unsuccessful locks and unlocks of <b>VSYSOPEN00000001</b> , where VSYSOPEN is interpreted as characters and 00000001 as hex value, of <b>all partitions</b> .
<b>LOCK TRACE,RESOURC*</b>	traces <b>all</b> unsuccessful locks and unlocks of resources, starting with the character string <b>RESOURC</b> of <b>all partitions</b> .
<b>LOCK TRACE=F6,RES*</b>	traces <b>all</b> unsuccessful locks and unlocks of resources, starting with the character string <b>RES</b> of <b>partition F4</b> .

For interpretation of the output please refer to the lock table description.

## Examples

```
lock trace=f4
AR 0015 1I40I  READY
F4 0025 LOCKTAB ENTRY
V0004F344 00324440 00000000 D9C5E2D6 E4D9C3C5 * à RESOURCE* R0004F344
V0004F354 60C5F1F0 11900001 0004F364 00000000 *-E10 ° 3Ã * R0004F354
F4 0025 OWNER ELEMENT
V00324440 00000000 002C0000 00011000 00000000 * * R0087F440
F4 0025 LOCKTAB ENTRY
V00324160 00324490 00000000 D9C5E2D6 E4D9C3C5 * à° RESOURCE* R0087F160
V00324170 60C5F1F1 11900001 00324420 00324400 *-E11 ° à à * R0087F170
F4 0025 OWNER ELEMENT
V00324490 00000000 002C0000 00011000 00000000 * * R0087F490
F4 0025 LOCKTAB ENTRY
V00324450 00324480 00324440 D9C5E2D6 E4D9C3C5 * àø à RESOURCE* R0087F450
V00324460 60C5F1F3 01900000 003244A0 00324420 *-E13 ° äff à * R0087F460
F4 0025 OWNER ELEMENT
V00324480 00000000 002C0001 00000000 00000000 * * R0087F480
F4 0025 REQUEST ELEMENT
V00324440 00000000 00250000 00000000 00600B18 * - * R0087F440
AR 0015 1I40I  READY
lock trace=off
AR 0015 1I40I  READY
```

```
lock trace=f4,resource-e10
AR 0015 1I40I  READY
F4 0025 LOCKTAB ENTRY
V0004F344 003245C0 00000000 D9C5E2D6 E4D9C3C5 * ā{ RESOURCE* R0004F344
V0004F354 60C5F1F0 11900001 0004F364 00000000 *-E10 ° 3Ã * R0004F354
F4 0025 OWNER ELEMENT
V003245C0 00000000 002C0000 00011000 00000000 * * R0087F5C0
F4 0025 LOCKTAB ENTRY
V0004F344 0004F404 003245E0 D9C5E2D6 E4D9C3C5 * 4 ā\RESOURCE* R0004F344
V0004F354 60C5F1F0 11900001 0004F364 00000000 *-E10 ° 3Ã * R0004F354
F4 0025 OWNER ELEMENT
V0004F404 00000000 002C0000 00011000 00000000 * * R0004F404
F4 0025 REQUEST ELEMENT
V003245E0 00000000 00250000 00000000 00600ABE * - ”* R0087F5E0
AR 0015 1I40I  READY
lock trace=off
AR 0015 1I40I  READY
```



---

## Channel Program Translation

**Note:** Whenever in this section (Channel Program Translation) a reference is made to a CCB (Channel Command Block), it also includes the IORB (Input/Output Request Block).

The supervisor must do the following before initiating an I/O operation (if not EXCP real):

- Translate format 1 CCWs into format 0 CCWs.
- Copy the CCB and the entire channel program into copy blocks in the supervisor.
- Translate the addresses used by the CCB and the channel program into real storage addresses and place these addresses into the copied CCB and channel program.
- Build IDALs (Indirect Data Address Lists) for all I/O areas which cross one or more page boundaries.
- Build IDALs for all I/O areas with a real address larger than 16MB.
- Fix all pages containing I/O areas in real storage for the duration of the I/O operation.

These functions are performed by the routine CCWTRANS. CCWTRANS is called by the channel scheduler every time a virtual-mode I/O request is made. For I/O requests from BTAM channel appendages this routine is entered at its entry point CCWTRBT2 (for further information, refer to -- Heading 'BTAMC' unknown --).

At the completion of an I/O operation, the routine CSWTRANS is called by the I/O interrupt handler. It must do the following:

- Retranslate the address of the last CCW pointed to by the CSW at channel end to its correct virtual address. This address is placed in the copied CCB.
- Free the data areas.
- Release the copy blocks used for the translation except the CCB copy block.
- Transfer the CCB information which has changed to the original CCB. If this is not possible (because the original CCB is not in real storage) indicate to the dispatcher that this must be done before the user task is given control again. In this case, the dispatcher calls a special routine (MOVECCB) to transfer the end of channel information from the copied CCB to the CCB in the user program.

## Translation Control and Copy Blocks

The following control and copy blocks are used to copy and translate a CCB and channel program for a virtual-mode I/O request:

- A translation control block (CCWTCB). This is a work and save area, located in the task control block (TCB) and used during translation.
- A CCB copy block. The user CCB and sense CCW (if any) are copied into this block. The CCB copy block also contains information about the copied and translated channel program.
- CCW copy blocks. Each block contains copy locations for up to 7 contiguous CCWs and queuing information.

- IDAL blocks used for building Indirect Data Address Lists for data areas which cross-page boundaries.
- Fix information blocks containing the page frame numbers of pages fixed for this request.

### **The Translation Control Block (CCWTCB)**

Because a translation request may be interrupted (by a page fault, wait), it is necessary that the translation routine be partially reenterable so that several requests may be handled simultaneously.

The CCWTCB is located in the work area of the task control block (TCB) of the requesting task. The other blocks are 72-byte blocks located at the end of the supervisor. They are dequeued from the free copy block queue (pointed to by AFCB) as needed, and enqueued again when they are no longer needed by the requesting task.

If the queue of free copy blocks is empty when a request for a copy block is made, one of the following actions will be taken:

- If the requesting task is the only one using the CCW translation routines, it will be canceled (not enough copy blocks available to ever satisfy the request).
- If the request is for a CCB copy block or if at least one request has been handled successfully, the requesting task is set copy block bound.

If no other task is complete, and if the request is not for a CCB copy block, the used copy blocks are freed and the task is set translation bound. When another translation has been successfully completed, the request will be started again from the beginning.

### **CCB Copy Blocks**

For each virtual-mode request one copy block is used to contain the copied CCB and its sense CCW, if any. The rest of the block contains control information about the translated program. Figure 57 on page 163 shows the layout of the CCB copy block.

If an Input/Output Request Block (IORB) is used for the request, bytes 0-15 (identical to a CCB) are set into the CCB copy block.

All the CCB copy blocks in use are queued in the queue pointed to by ACCBB. Each CCB copy block is also individually pointed to by a field in the request's TCB. After translation, the address of the copied CCB is placed in the channel queue. Figure 57 on page 163 shows the mutual and external relationships of the CCB copy blocks.

	0	1	2	3	4	5	6	7	
0	CCBCNT		CCB COM1	res- erved	CCB STA1	CCB STA2	CCB CLS*	CCB LNO	↑ Copied CCB ↓
8	CCBCCW Address of first CCW				CCBBY3	CCBCSWW			
16	CCBSENS Sense CCW if any								
24	TID TASKID		CCB Flag**	Unused	CCBVA Virtual address of CCB				
32	CCBACB Address of first CCW copy block in channel program with lowest VBA				CCBICB Address of first IDAL block in channel program				
40	CCBXINF (Fix information) Real page numbers of TFIxed pages								
64	CCBXPTR Address of additional fix information block				*** X'A0'	CCBNEXT Address of next CCB copy block			

Figure 57. CCB Copy Block

- \* - Bit 2 is set (X'20') to indicate copied CCB
- \*\* - Legend CCBFLAG:

**Bits Description**

- 0: Indicates that CCW-translation of this request is complete; indicator is set before I/O request is enqueued in channel queue.
- 1: Indicates that control has been transferred to TFIx routine at least once during CCW translation; if 0, scan through CCBXINF for freeing pages is skipped; indicator is set immediately before control is passed to TFIx routine.
- 2: Reserved.
- 3: Reserved (former BTAM request)
- 4: Indicates that the channel program is valid for fast CCW translation (CCWs are contiguous, not a system task request with an I/O area in the SVA).
- 5: Indicates that this CCB copy block is on the saved CCB queue.
- 6: Indicates that the pages containing I/O areas for this channel program require fixing.
- 7: Reserved.

- \*\*\* - 'Block in use' indicator

**CCW Copy Blocks**

Each CCW copy block consists of 7 copy locations and 16 bytes for pointers and inserted TIC commands. The layout of a CCW copy block is shown in Figure 59 on page 165.

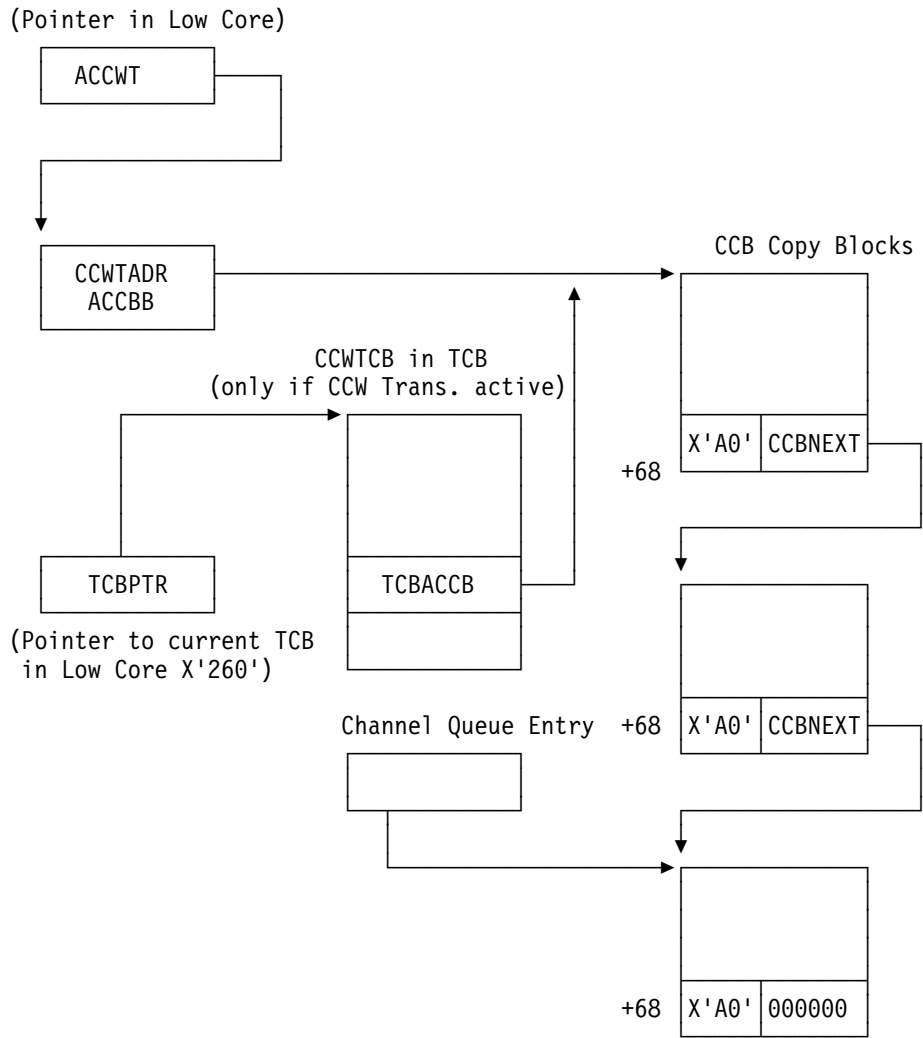


Figure 58. Locating CCB Copy Blocks



	0	1	2	3	4	5	6	7
0	1st Copy location for CCW							
8	2nd Copy location for CCW							
16	3rd Copy location for CCW							
24	4th Copy location for CCW							
32	5th Copy location for CCW							
40	6th Copy location for CCW							
48	7th Copy location for CCW							
56	X'A0' *	X'000000'			Virtual address of first CCW in copy block (VBA)			
64	X'A8' **	X'000000'			*** X'A0'	Addr. of next CCW copy block in chain (ANB)		

Figure 59. CCW Copy Block

**Notes:**

1. \* X'A0' indicates the end of the CCW copy locations in the block. It is replaced by a TIC (Transfer in channel command) if the 7th copy location contains a copied CCW with data- or command chaining. Bytes 57-59 will then point to the copy location of the CCW following the CCW in the 7th copy location. Bytes 56-59 will not be changed if the CCW in the 7th copy location is a TIC.
2. \*\* X'A8' indicates the last 8-byte entry in the block. It is replaced by a TIC if the CCW in the 7th copy location is a status modifier CCW. Bytes 65-67 will then point to the copy location of the second CCW following the status modifier CCW.

The CCW copy blocks for a translation are queued in order of increasing VBAs (see Figure 59) with the lowest one being pointed to by the field CCBACB in the CCB copy block. Figure 60 on page 166 shows the relation of CCW copy blocks to one another.

3. \*\*\* X'A0' 'Copy block in use' indicator.

## IDAL Blocks

CCWs whose data areas cross 4K boundaries must have an IDAL (Indirect Data Address List) in the copied channel program. If a ESA supervisor operates on a machine with more than 16MB real storage, CCWs will have always IDALs in the copied channel program.

In both cases, the CCW is changed to show that an IDAL is used (bit 37 of the copied CCW is set) and the address of the IDAL is placed in the data address of the CCW. The IDAL pointed to contains one entry for the beginning of the data area and one entry for each 2K boundary crossed.

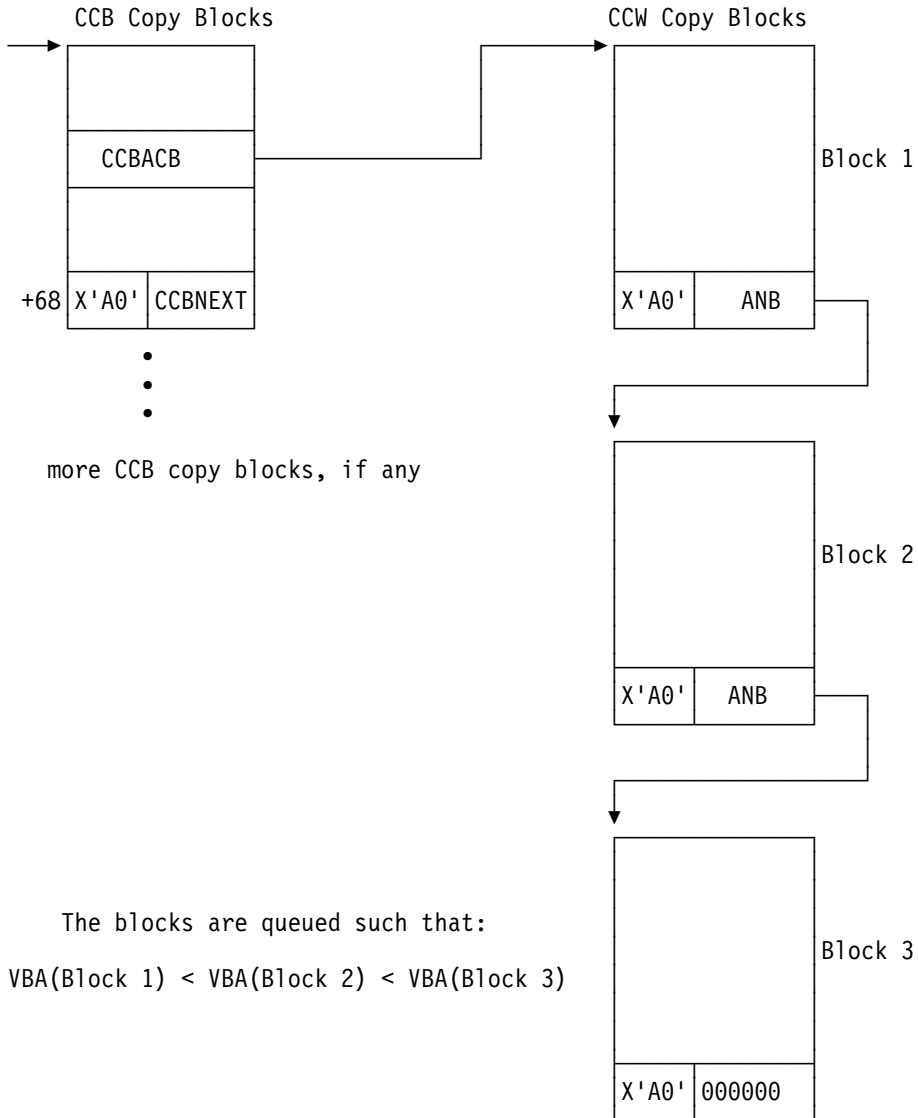


Figure 60. Locating CCW Copy Blocks

An IDAL must be located in consecutive copy block locations, so that if an IDAL cannot fit into the last block in the queue (the count in IDALCNT is less than the number required) a new block must be enqueued. For I/O areas with a length of less than 32KB a single copy block is dechain as IDAL block with 17 locations for Indirect Data Address Words (IDAWs). If the area is larger than 32KB two consecutive copy blocks are dechain from the free copy block queue. This double block has 33 locations for IDAWs.

After an I/O area has been TFIxed in real storage, the addresses in the IDAL are translated to point to the correct real storage locations (the begin address of the I/O area and the begin address of the page frames for the rest of the I/O area, or for a read-backward command, the end address of the I/O area, and the end address of the page frames).

Each IDAL is pointed to by the CCW which references it. In addition, the IDAL blocks are queued with the first one being pointed to by the field CCBICB in the CCB copy block. Figure 61 shows the relation between the IDAL blocks and the other blocks.

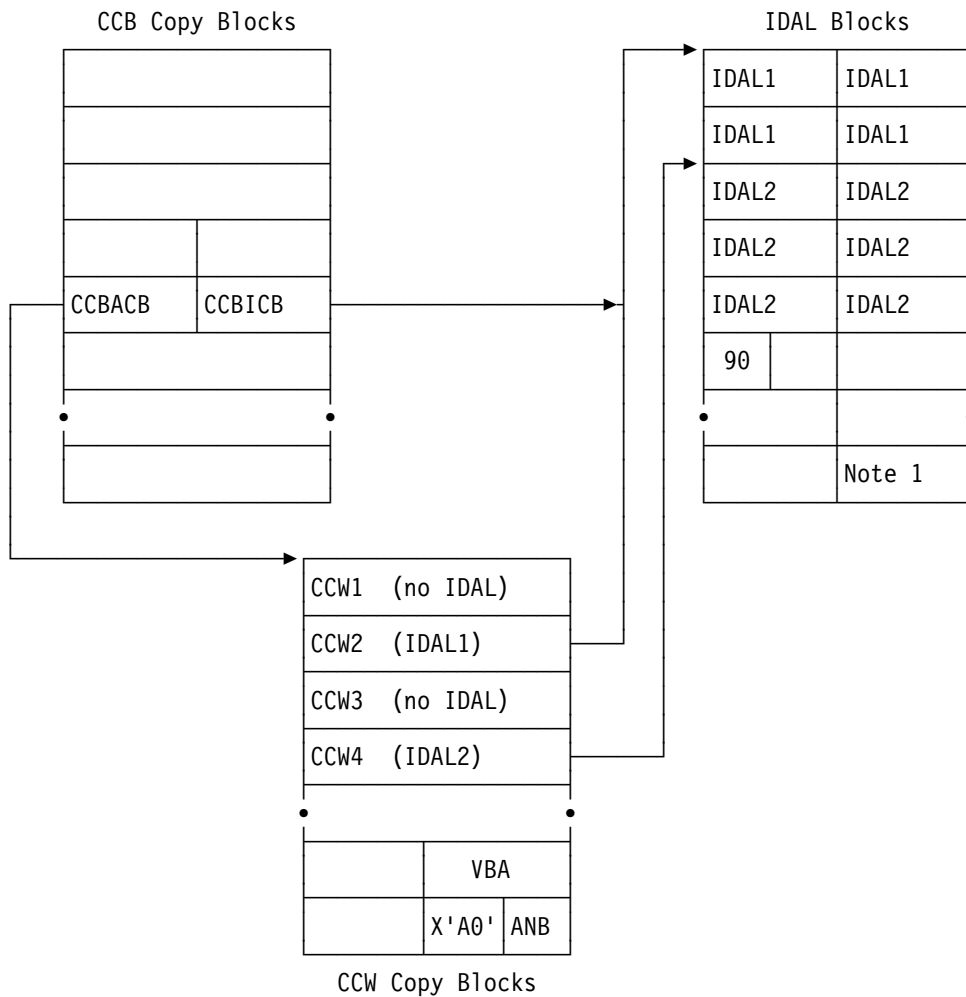
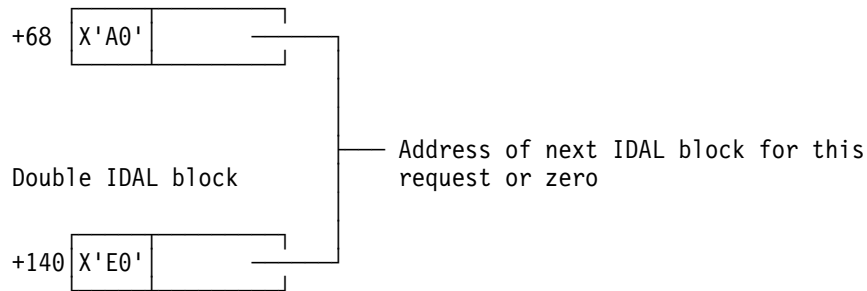


Figure 61. Relation of IDAL Blocks to other Blocks

## Notes:

1.

Single IDAL block



the contents of X'C0' being:

X'A0' Block in use

X'40' Double copy block

- The X'90 in the first byte of the 11th IDAW indicates the end of the IDAWs for the block. In this case, the IDALCNT field in the CCWTCB would show seven free copy locations.
- The data area of CCW2 crosses three 2K boundaries (may be up to 8KB) and the data area of CCW4 crosses five 2K boundaries (may be up to 12KB).

## Fix Information Blocks

In order to keep track of which page frames have been TFIXed for a request, the real page frame numbers of the pages fixed are kept in the copied CCB at label CCBXINF. If more than six pages have to be TFIXed for the I/O request, additional copy blocks are used. They are queued with the first one being pointed to by CCBXPTR in the copied CCB.

A page used more than once by a request is only TFIXed once.

## Copying and Translating Channel Programs

User channel programs are copied into the copy blocks described in the previous section by the routine CCWTRANS (entered at CCWTRBT2 for BTAM channel appendage I/O request).

By way of initialization, the following is done before the actual copying and translation is begun:

- The CCWTCB for the requesting task is initialized. As part of the initialization procedure, the TCB pointers to the two special command lists for the device are filled in (see Figure 62 on page 169).
- Two copy blocks are dequeued from the free copy block queue for the CCB copy block and the first CCW copy block.
- The CCB is copied and initialized so that the CCW address points to the first location in the first CCW block. The VBA in the first CCW copy block is set to the virtual address of the CCW the virtual CCB is pointing to (which is the virtual address of the first CCW to be executed).
- If a sense CCW was present, it is also copied into the CCB copy block and its data areas are TFIXed in real storage (unless it crosses a 2K boundary, in which case an IDAL is built), and the address is translated.

The channel program is then copied and any necessary IDALs are built. The channel programs translated can be divided into three classes according to the types of commands they contain. They are described in the following order:

1. Channel Programs without TIC or Status Modifier Commands.
2. Channel Programs with TIC Commands.
3. Channel Programs with Status Modifier Commands.

A schematic representation of channel program translation is shown in Figure 63 on page 171.

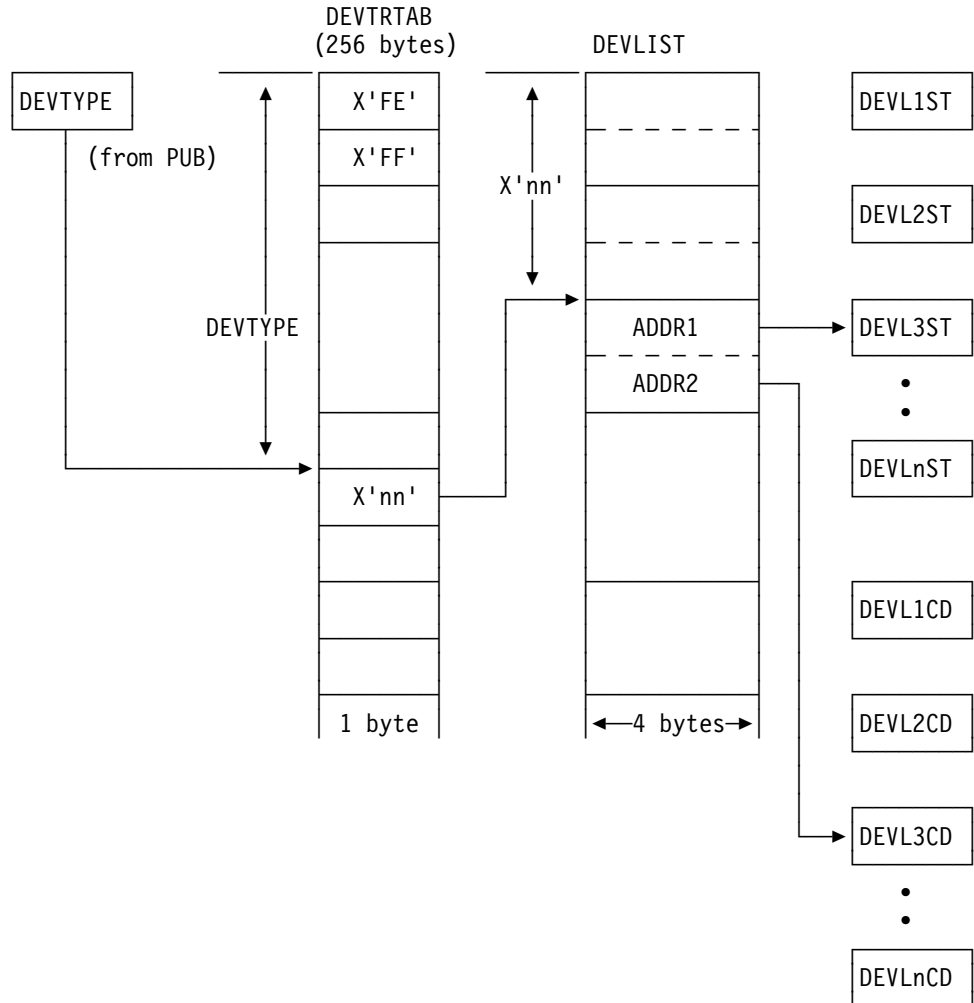


Figure 62. Initializing Special Command List Pointers in CCWTCB

DEVTYPE: Device type code from PUB  
DEVTRTAB: Entries:  
X'FF' = Unsupported device.  
X'FE' = Device does not support status modifier commands or control commands with data area.  
X'nn' = Displacement to entry in DEVLIST if device supports status modifier commands and/or control commands with data area.

DEVLIST: List of pointers to the special command lists. The two entries (if any) for the device on which the I/O is requested are moved to the TCB when this is initialized.

DEVLnST: Status modifier command list for device type n.

DEVLnCD: Control command with data area list for device type n (see note below).

**Note:** DEVLnST and DEVLnCD are bit strings. When a CCW is copied, the command code is used to refer to a bit in these strings. By testing this referred bit it is determined whether a CCW is a status modifier command or a control command with data area, or does not belong to these categories.

### **Copying Channel Programs without TIC or Status Modifier Commands**

The first CCW in a channel program is always copied into the first copy location pointed to by the copied CCB. If command chaining or data chaining is specified in the CCW the following chained CCWs are copied into successive copy locations.

If a program of chained CCWs should contain 8 or more commands, a new CCW copy block must be used. The eighth copy location of the first copy block is then converted into a TIC command pointing to the first location of the next copy block. The VBA of the next copy block is set to the virtual address of the eighth chained CCW.

Figure 64 on page 173 is an example of a copied channel program containing 11 chained CCWs.

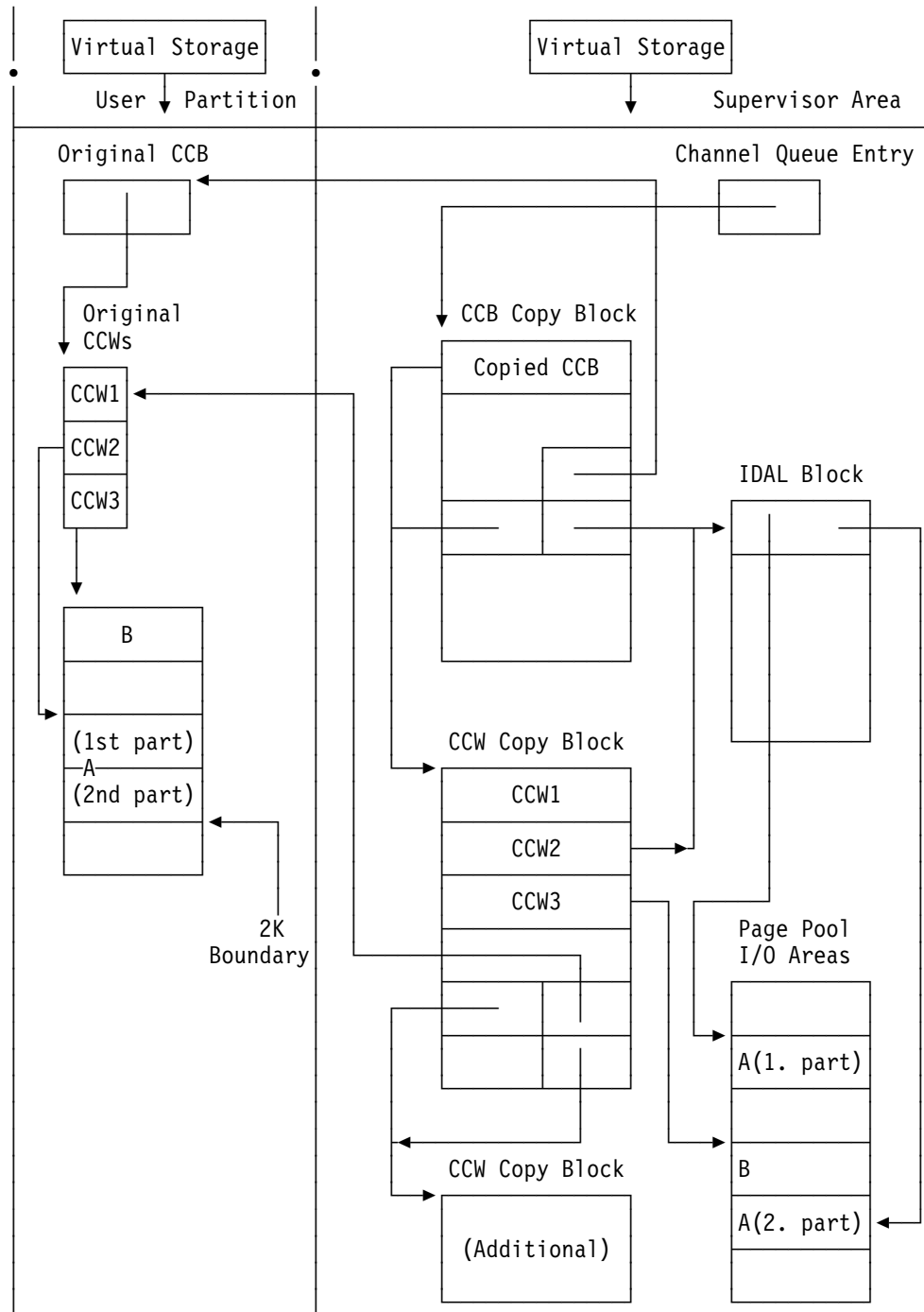


Figure 63. Schematic Representation of Channel Program Translation

### Copying Programs Containing TIC Commands but no Status Modifier Commands

A TIC command (transfer in channel) command is, when encountered, copied into the next copy location just as any other chained command is. Although a TIC is 8 bytes long, only the first 4 bytes have any meaning (the command code and transfer address). The second four bytes of the copied TIC are set to zero. These bytes are used as a chain pointer for TICs which follow status modifier commands (refer to the section "Copying Status Modifier Commands"). The command code of a copied TIC is set to X'08' (standard user TIC).

The virtual storage location pointed to by the TIC command must be mapped into a location in the copied channel program. This mapped location is then placed in the copied TIC (unless the copied TIC is the first location of a copy block, in which case the address is placed in the end-of-block TIC (eighth copy location of the previous copy block) and used as the copy location for the CCW pointed to by the TIC. The mapped location is determined in the following way:

- If the CCW pointed to by the TIC command has a copy location in an existing copy block (that is, there is a block such that the virtual CCW address lies between the block's VBA and the block's VBA+56), place the location thus found in the TIC and copy the CCW in the location if it is free. If the location is not free, go to the translation termination routines. Figure 65 on page 174 is an example of a TIC which points to an already existing copy location.
- If there is no existing copy location, a new CCW copy block must be enqueued. The new block is enqueued at either end of the existing queue or between two existing blocks, depending upon where the virtual address in the TIC is in relation to the VBAs of the existing blocks. Figure 60 on page 166 shows how a new CCW copy block is queued to provide a copy location for a CCW pointed to by a TIC. Once enqueued, the VBA of the new copy block must be determined. If at all possible, the new block will be aligned to the one either above or below it (the VBA is 56 greater than the VBA of the lower block or 56 less than the VBA of the upper block). This is only possible if the address pointed to by the TIC lies within one of the ranges (that is, is less than 56 below the VBA of the above block or less than 112 above the VBA of the block chained below). If possible to align to both blocks the alignment is made to the lower block. Considering the example in Figure 66 on page 175 again it is copied in the fourth copy location.
- If it is possible to align the new block to both the upper and lower blocks but not to both at the same time (the difference between the VBAs of the two blocks is less than 112), a short block must be created by moving the end-of-block indicators to the copy location following the last logical copy locations. Figure 67 on page 176 shows how a short block is enqueued.
- If no alignment of the new block with either of its neighbors is possible, the VBA of the new block is made equal to the virtual address pointed to by the TIC and the first copy location in the block is used. Figure 68 on page 177 shows such a copy block being enqueued.



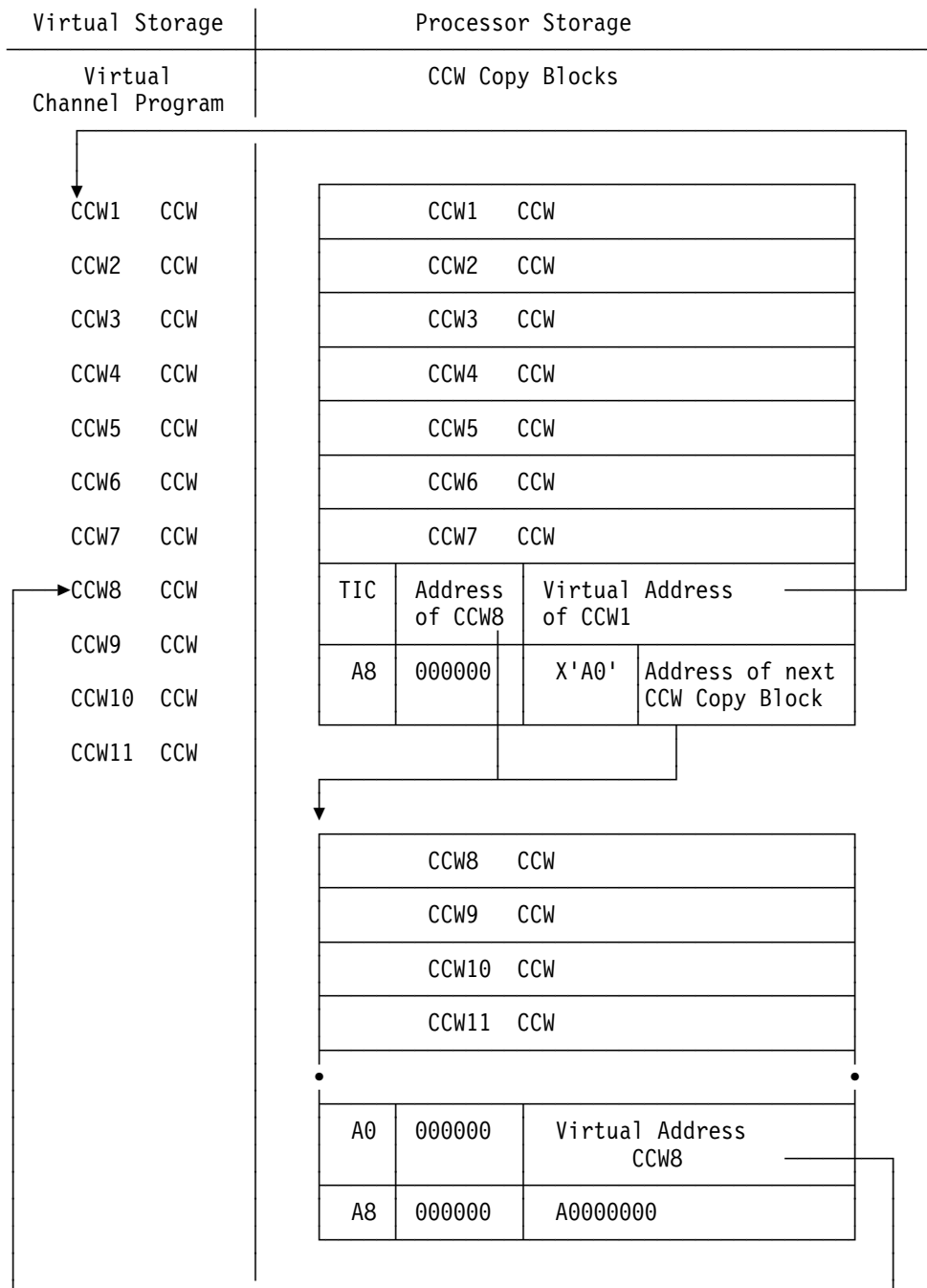


Figure 64. CCW Translation for a Channel Program. Without TIC or Status Modifier Commands.

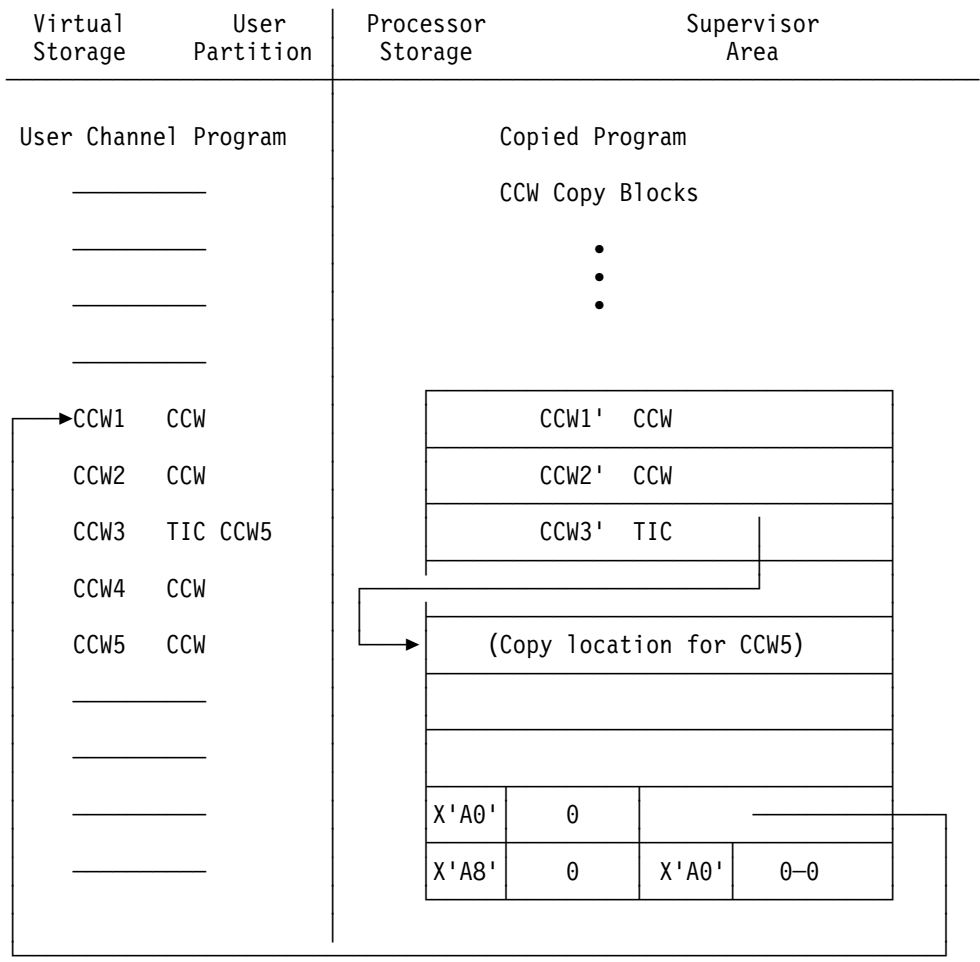


Figure 65. Copy Location for a CCW Pointed to by a TIC. If location is in already used copy block.

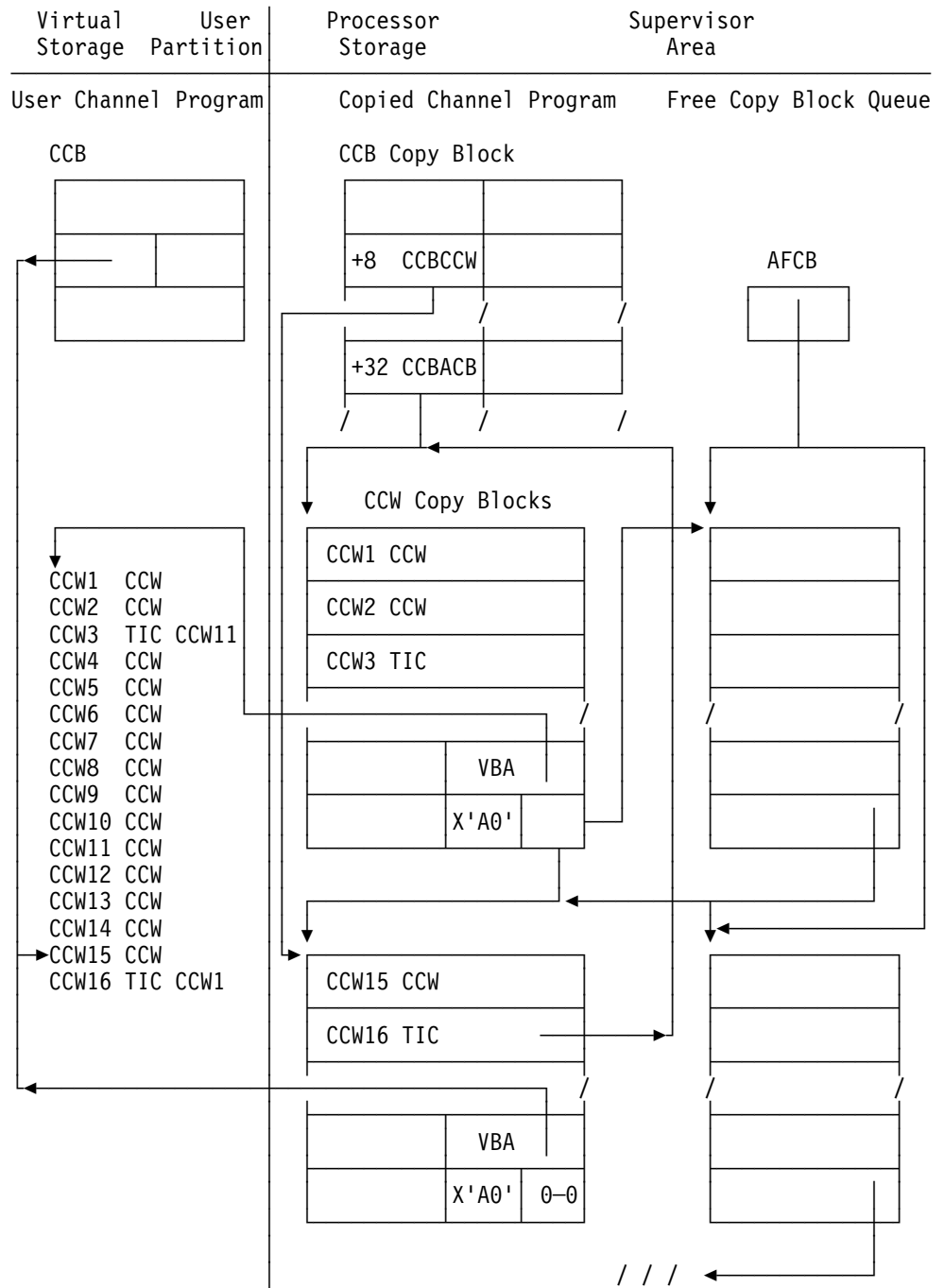


Figure 66. Enqueuing a New Copy Block. To the correct location in the CCW copy block chain to handle a CCW pointed to by a TIC (see Note 1).

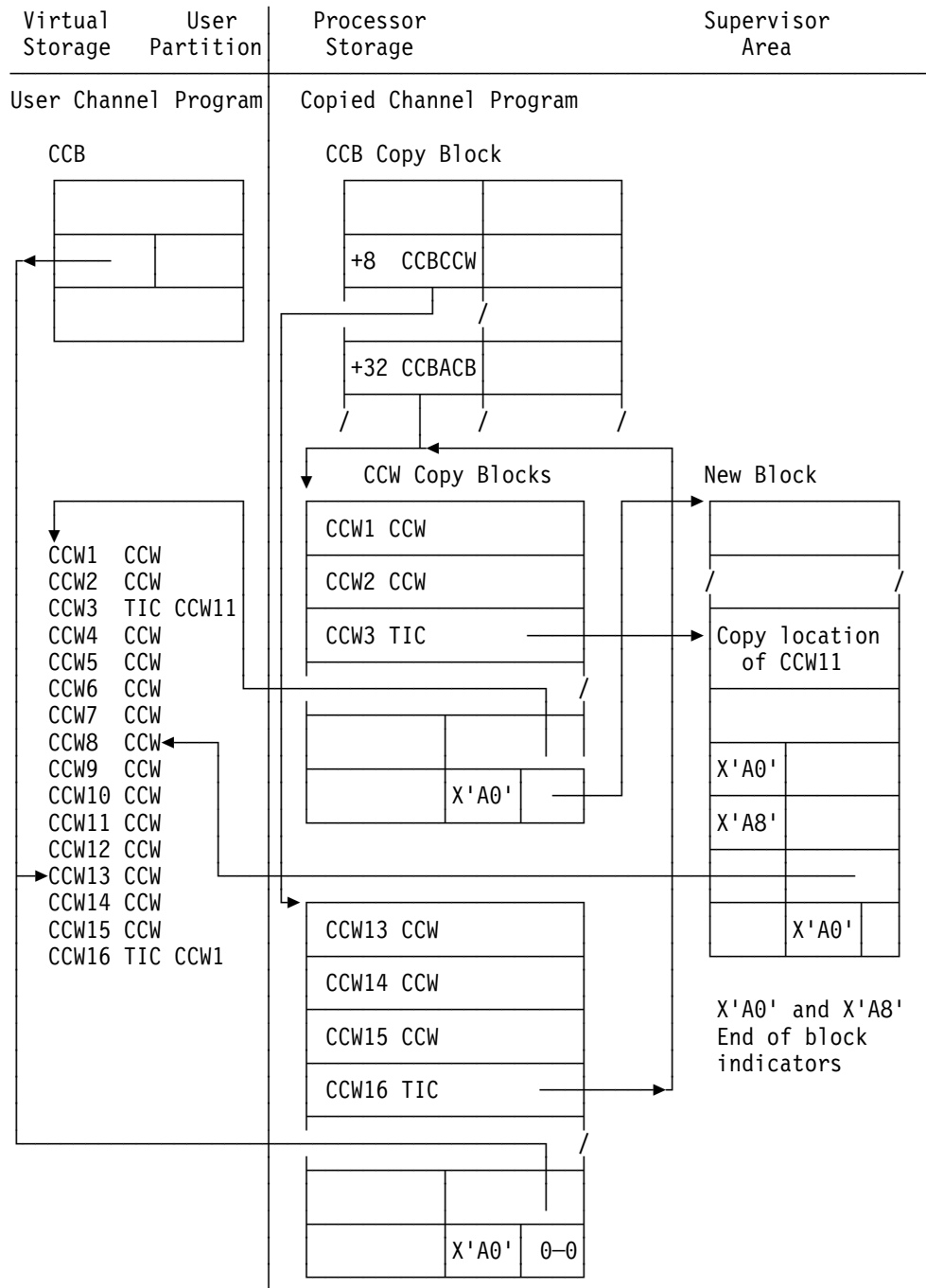


Figure 67. CCW Copy Block Queuing. Requiring the creation of a "short" block to maintain alignment (see Note 2).

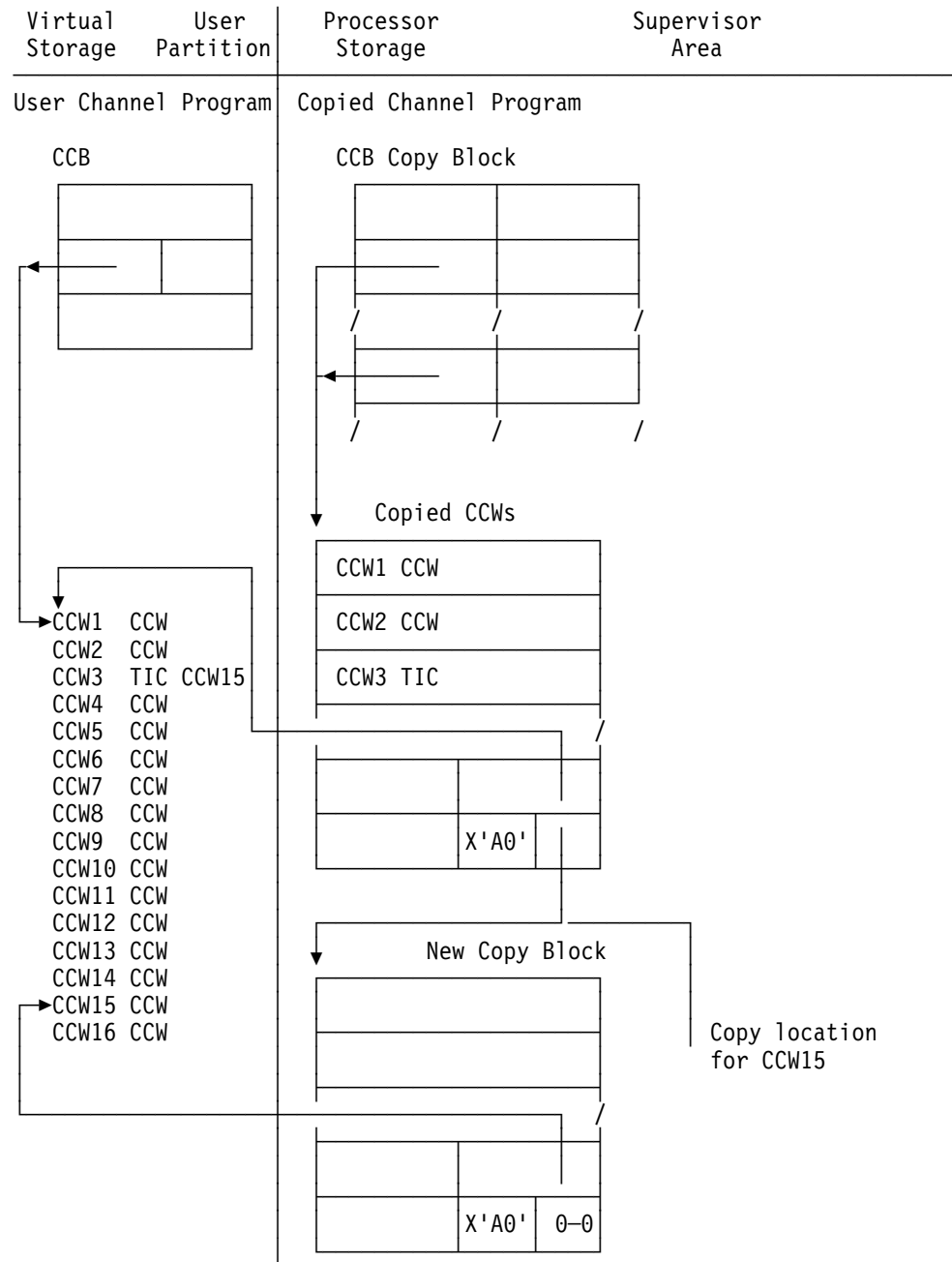


Figure 68. Enqueueing New Copy Block to Existing Block. Because the copy block cannot be aligned, CCW is too far removed from VBA of any existing block (see Note 3).

**Notes:**

1.

**Problem** CCW3 has just been copied. The problem is to find the copy location for CCW11.

**Solution** Free copy block is queued between A and B because the address used by the TIC at CCW3 lies between the VBA for A and the VBA for B. The solid line shows the condition before the new block is enqueued and the dotted lines the condition afterwards.

Once enqueued the VBA in the newly enqueued block will point to CCW8 (the block is aligned to the next lower block) and the TIC in CCW3 will point to the fourth copy location in the new block. Copying will then continue with CCW11 being copied into that location.

2.

**Problem** CCW3 has just been copied and the copy block for CCW11 has been enqueued. The problem is to align the block.

**Solution** Make the new block a 'short' block in that the end of block indicators are moved to the copy position following that for CCW12.

3.

**Problem** CCW3 has just been copied and it is necessary to find a copy location for CCW16, the next CCW copied.

**Solution** Enqueue a new copy block behind the first one and use the first copy location for CCW16 because it is impossible to align the new block to an existing block.

## Copying Status Modifier Commands

Status modifier commands may transfer control to either of the next two following CCWs depending upon the result of the status modifier's operation. If, for example, a SEARCH command is unsuccessful, control is transferred to following CCW. If it is successful, on the other hand, the following CCW is skipped and control is passed to the second following command.

Consider the following chain of commands:

```
    READ
    READ
    SEEK
    SEARCH
    TIC   A
    READ
    READ
A   WRITE
    WRITE
    SEARCH
    TIC   B
    READ
    READ
B   READ
    READ
```

If the first SEARCH in this program is successful, no branch is taken as the TIC command is skipped. If the SEARCH is not successful the chained commands beginning at A are executed. The same is true when the second SEARCH is encountered. This can be done any number of times in a program. Since a program is copied as it is executed, the presence of status modifier commands makes it necessary to take several passes through a program in order to cover all the possible branches.

In the first pass through a program, a TIC following a status modifier command is copied but otherwise ignored (unless the status modifier is copied into the last copy location of a copy block). The TICs thus encountered are queued in a line pointed to by LINEPTR in the TCB (the queuing addresses are in the second 4 bytes of the copied TICs). Figure 69 on page 181 shows a program with status modifier commands after the first pass has been made a copying it.

If a status modifier command happens to be copied into the last copy location of the block, an entry in a different queue is made. This contains as entries the last locations of blocks where a status modifier command is copied into the last copy location. The first entry in the queue is pointed to by BENDPTR in the TCB. The queuing addresses are in bytes 1-3 of the queue elements (last location of the CCW copy blocks concerned). Copying continues with the first CCW following the status modifier command being copied into the first location of the next queued copy block, and, if chained, copying continues with the following command. If, as is usually the case, the first command after the status modifier command is a TIC, the branch taken by the TIC command is copied. Figure 70 on page 182 shows a program with a status modifier command in the last copy position.

As soon as an end is reached in copying a program (a command without data or command chaining is copied or a copy location for a command is already filled) the program checks to see if there are any members in the queue pointed to by

LINEPTR or BENDPTR. The members of these queues are handled one at a time. See Figure 70 on page 182 to Figure 72 on page 184.

**Note:** LINEPTR and BENDPTR entries can be created while others are being handled. Translation is complete when both LINEPTR and BENDPTR are zero (that is, no more entries in either queue).

### **Translating Data Addresses and Page Fixing**

Parallel to the copying of a channel program, the pages containing the data areas for the various CCWs are TFIXed in real storage and the virtual addresses of the data areas are translated into real addresses.

IDALs are first built using the virtual addresses of the beginning of the data area and the 2K boundaries. When the individual pages are TFIXed in real storage these addresses are replaced with the correct real addresses. Figure 73 on page 185 shows an IDAL built for a data area both before and after the pages have been TFIXed. Figure 74 on page 186 shows how the IDAL looks if the command is a read backward command.



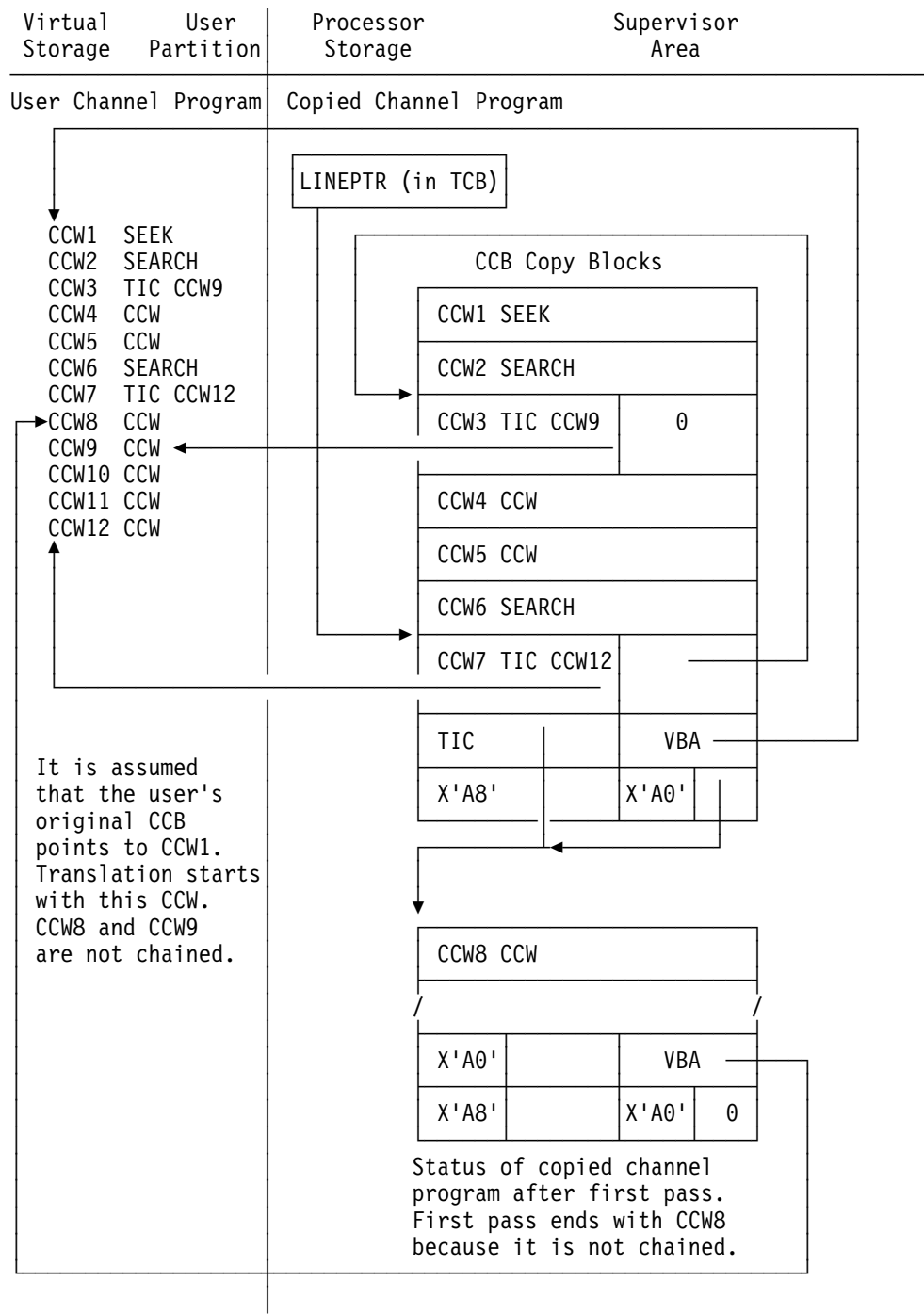
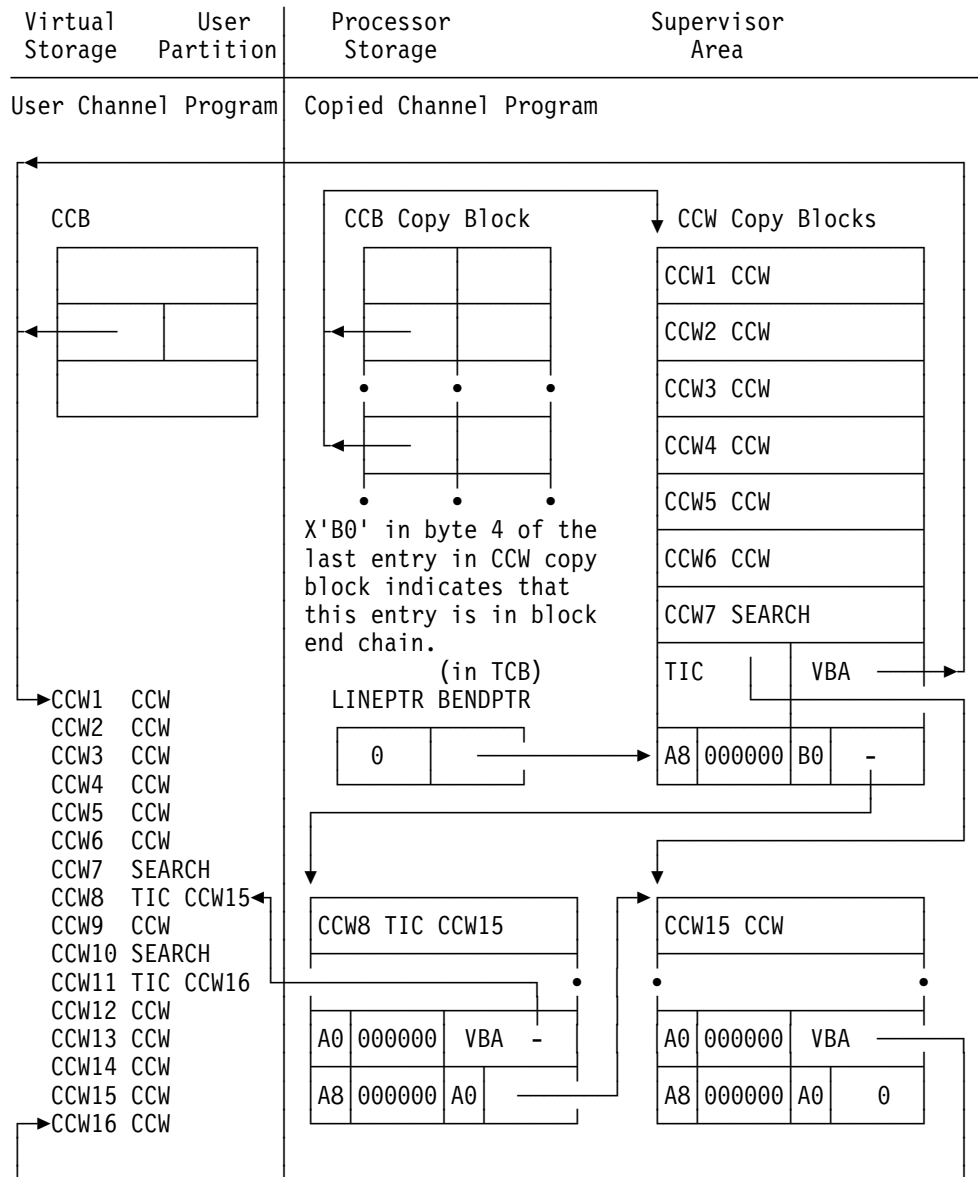


Figure 69. Channel Program. Containing status modifier commands after its first copying path has been made.



It is assumed that CCW14, CCW15 and CCW16 are not chained.

That status modifier command CCW7 is copied into the 7th copy position necessitating an entry into the BENDPTR queue. The first pass ends when CCW15 is copied, because this CCW is not chained.

Figure 70. Channel Program. Containing status modifier commands after its first copying path has been completed.



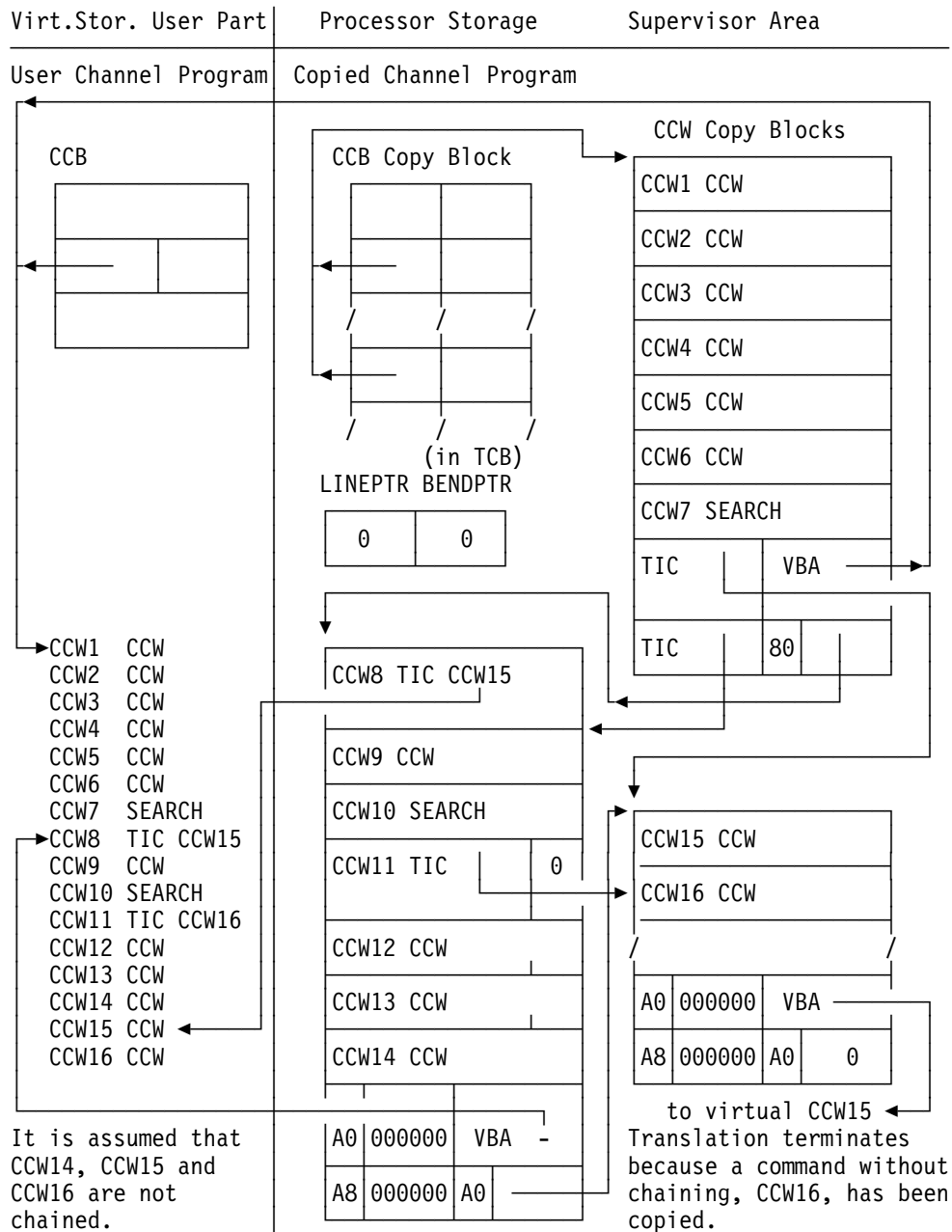


Figure 72. Channel Program. Containing status modifier commands after completion of translation.

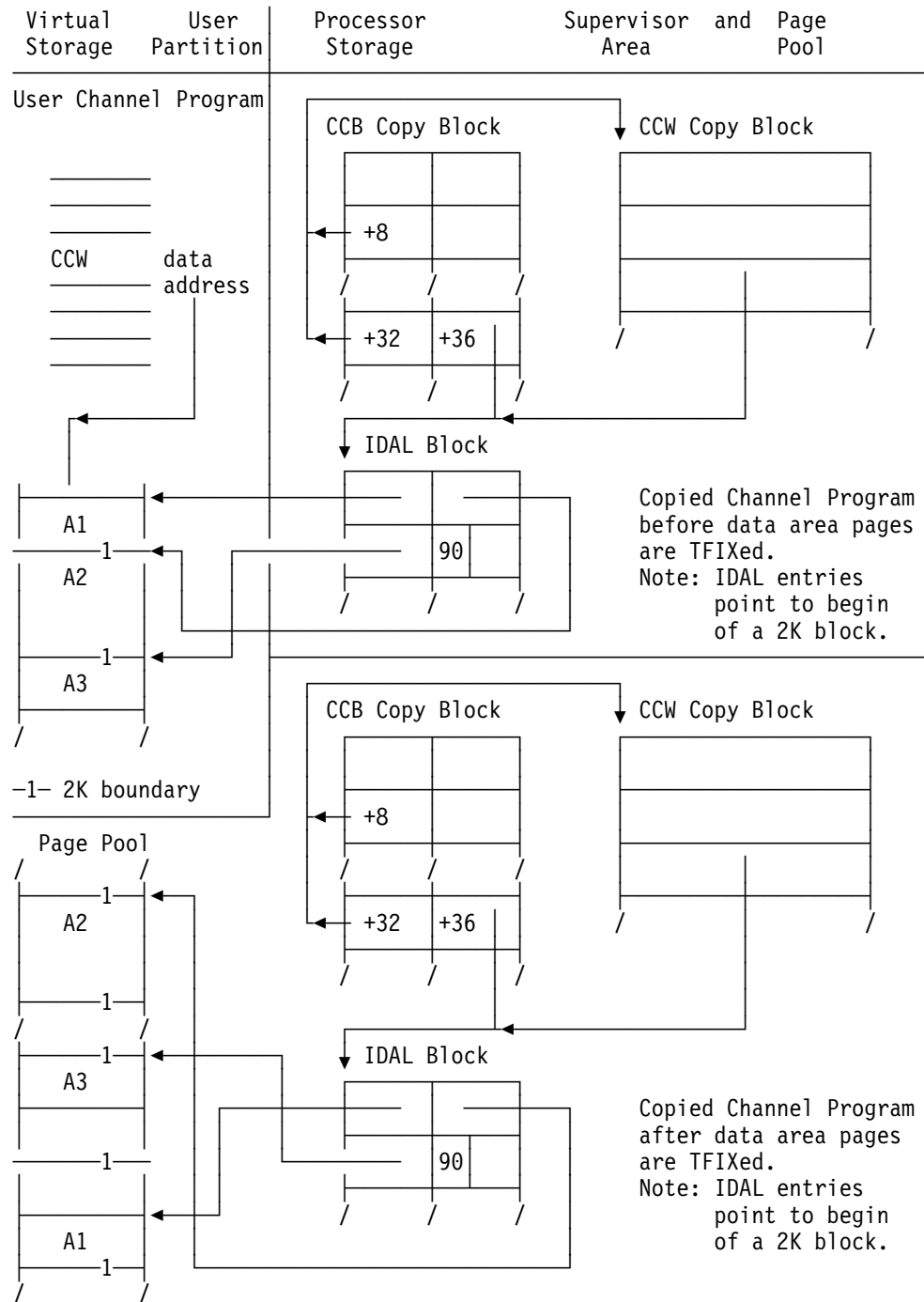


Figure 73. Copied CCW. Requiring an IDAL to be Built (normal READ or WRITE command)

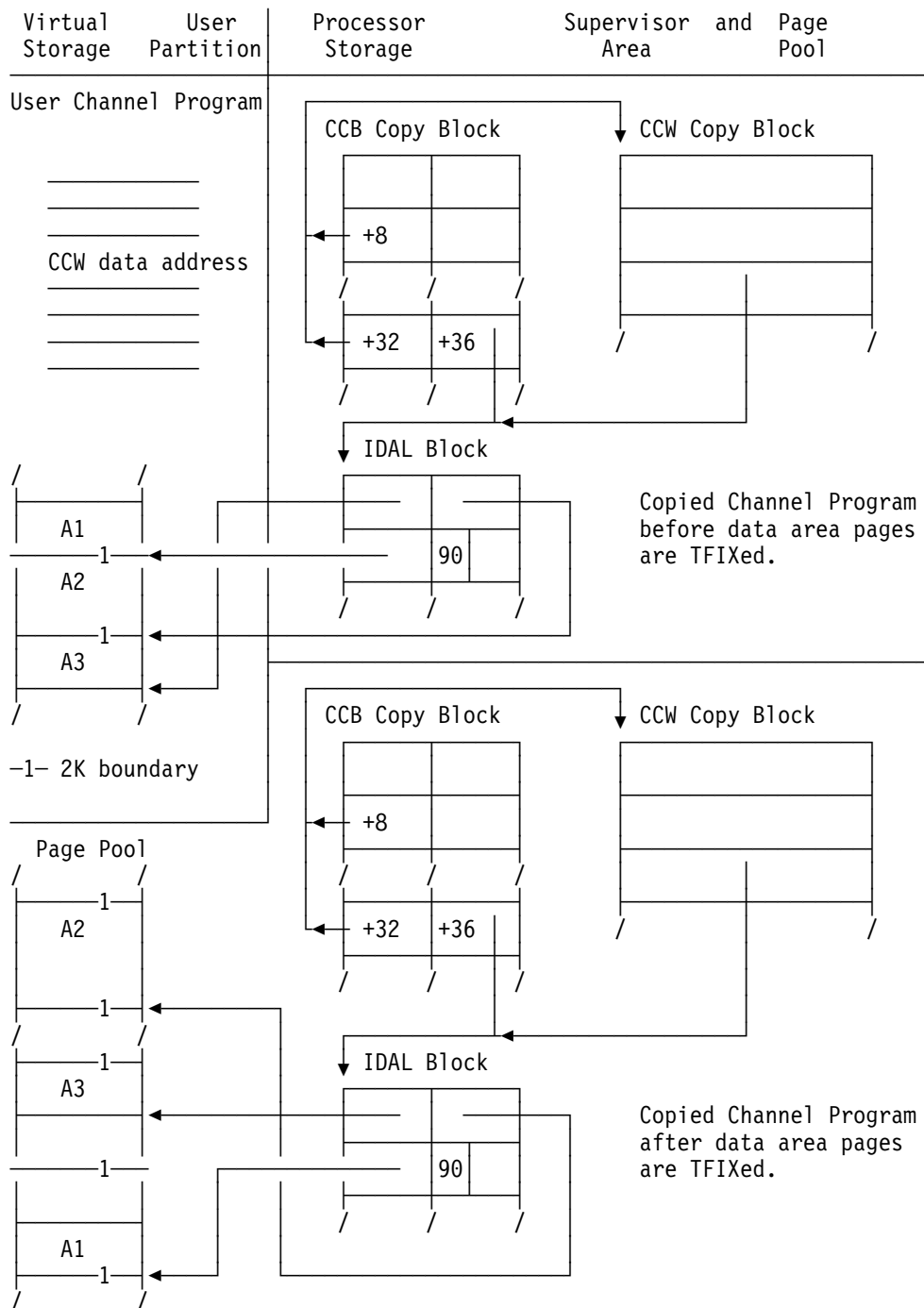


Figure 74. Copied CCW. Requiring an IDAL to be Built (READ Backward Command)







---

# Page Management

---

## General

### Introduction into Page Management

The page management is responsible for the management of the data set containing the virtual address and data space(s), for the allocation of the processor real storage to parts of the virtual space being requested and for the related replacement strategy. The unit of logical storage is the PAGE, the data set is called PAGE DATA SET (PDS). The real storage area containing a page is called a PAGE FRAME.

A page management function satisfies those processor requests created by addressing a valid logical area not yet assigned to and located in real storage (PAGE FAULTS). The related page is in disconnected state and its copy - if valid - has to be read from the PDS into a selected page frame. The page has thereafter addressable state. However, if there is no free page frame, at first the page currently located in the selected page frame has to be saved onto the PDS before the frame can be used by the new page. The state of the saved page is changed from addressable into disconnected.

A further function provides the capability to FIX a page in the real storage. This function is required for the I/O subsystem which operates on real storage (page frames) only. Because of performance considerations the fixing can be also desirable for frequently used address ranges.

Another function allows the user to control the paging environment by its own services. These services are implemented for the various subsystems to allow an optimization of the 'page' resources.

As seen, the total page management can be subdivided into the following main parts:

- Page handling support
  - Page fault handling together with the page selection algorithm
  - Page out handling
  - Pseudo-Page Fault handling
  - SVC services concerning page state (SVC106, SVC109/121)
  - Subroutine service concerning page state (INVPAGE)
- FIX / FREE support
  - TFIX / TFREE services for the I/O subsystem, FETCH and SVC44
  - SVC services for user PFIX / PFREE (SVC67/121, SVC68/121)
  - subroutine service for SVA PFIX / PFREE (SVAFX2ND)
  - SVC services for allocation of real storage with SVC54 and SVC55
  - SVC service for CHECKPOINT / RESTART (SVC74)
- Page handling by user
  - PHO capability (Page Fault Overlap) with SVC71/121
  - SVC services concerning page-in, release page and forced page out (SVC85/121, SVC86/121, SVC87/121)
  - VIO (Virtual I/O ) support.
- Load-Levelling:

- Deactivation, Reactivation routines
- Teleprocessing Balancing (SVC88/89)

All page management services are called and executed in AMODE(31), with the following exceptions:

- Parameter list handling routines of PFI, PFREE, RELPAG, FCEPGOUT and PAGEIN, which are executed in AMODE(requestor)
- Deactivation, Reactivation routines are called and executed in AMODE(24)
- SCANPGT service is called and executed in AMODE(24)
- All SVC services are invoked with AMODE(24)
- All SVC services are executed in AMODE(31) with the following exceptions:
  - SVC 71 (SETPFA)
  - SVC 88 (TPIN)
  - SVC 89 (TPOUT)

## Support of Processor Storage above 2GB.

Processor storage above 2 GB is used by the page manager for paging purposes. These page frames cannot be used transparently, though. There are a few restrictions.

Page frames must be assigned below 2 GB to

- PFIed pages
- TFIed pages
- Real partition pages
- Prefix area

A real address below 2 GB must be returned for

- LRA

### 64-bit Page Frame Queues

The page frames below and above 2GB are organized in separate PFTE queues because of their selective use. A PFTE - *page frame table entry* - represents a page frame. There are two queues, IPFQ and IPFQ64 for invalid 31-bit page frames, respectively invalid 64-bit page frames. There are two page frame selection queues PSQ and PSQ64. The main reason is: To keep search loops for available 31-bit page frames short after a PFI or TFI request. Also page selection after a page fault is faster with separate PFTE queues.

### 64-bit Page I/O

Page I/O for page-out and page-in can be done with 64-bit page frames. The page manager builds Format 1 CCW chains. IOS does not modify the channel programs and executes the Format 1 CCWs.

### 64-bit Addressing

64-bit addressing mode is required for managing storage above 2GB. Since the Page Frame Table is allocated at the high end of processor storage this mode is needed when the page frame table entries or the page frames are manipulated or inspected. Therefore most of the page manager code executes in 64-bit addressing mode. Also services like PFI/TFI have to execute partly in 64-bit mode.

Affected are routines and subroutines of

- Page fault handler (SGPMR),
- General page management routines (SGPDATA),
- Prefix allocation (SGPPMT),
- Page fixing services (SGPFIX),
- Allocation and fixing of real space areas (SGPREAL),
- VIO services (SGPSVC),
- Page fault optimization services (SGPOPT).

### **Increased Storage Requirement for PMR Tables**

The size of segment tables and page tables have doubled in z/Architecture. About 2.4 MB are required now for page manager control blocks to support a private area of 1 GB.

The page frame table has doubled in size because of the 64-bit chaining pointers for the page frame table entries. The page frame table needs 16 MB of processor storage for 1 GB of processor storage.

## **Description of Parallel Page I/O**

Parallel page I/O is done by overlapping the page I/O operations for separate page-data-set devices. Therefore, parallel page I/O requires a multiple extent page-data-set, at best each extent distributed on a separate device but at least two extents on two devices.

For every page-data-set device, there is one page-in queue per static partition (inclusive system partition), one page-in queue per dynamic class (that means, page faults for dynamic partitions belonging to the same class are queued in one queue) and one page-out queue.

The I/O operations are controlled by a system task, the so called PMR-task. The page-data-set devices are serviced in wrap-around mode. The PMR-task tries to start an I/O request on each device as long as requests are pending and not yet started. Thereafter, the PMR-task waits for completion of at least one I/O.

However, before the page-fault request is enqueued it is checked whether the request can be serviced without any I/O. If so, the request is handled under the requesting task without any activation of the PMR-task.

## **Handling of Address Spaces**

VSE/AF supports *n* virtual address spaces, each up to 2GB. Each address space is separated into a private addressable area and a shared addressable area. The shared area is unique in the system. Programs and data used in any address space must be located in the shared area (for example, supervisor routines, SVA programs, control blocks in the system GETVIS area). The sum of all private areas and the shared area is restricted to 90GB, an arbitrary limitation. (The real limit is the size of the maximal 15 page-data-set extents/devices).

The address translation, as defined by the z/Architecture (see *Principles of Operation*, SA22-7832) is done via *Segment Table Entry* (STE) and *Page Table Entry* (PTE). Each STE addresses a list of contiguous PTEs which describe a logical address range of 1MB, one segment. The architected page size is 4KB. The STE points to a page table of 256 entries.

Region tables are not needed by z/VSE, because the virtual address spaces are restricted to 2 GB in size.

The different address spaces are represented by different segment tables (SGT) and are managed by *Space Control Blocks* (SCB). The shared area is addressable via any valid segment table (that means: it is part of any address space), but nevertheless the shared area is represented by an extra segment table with an invalid private area (partitions allocated in shared area are dispatched with this segment table). Each private address area is only addressable via one unique segment table.

The virtual address range of one addressable area (private or shared) can be thought as one contiguous and linear area. This area is represented by a list of Page Table Entries (PTE). Each entry is associated to an unique virtual address range (page) within the private/shared area and to a unique block on an external storage medium (these blocks build the Page Data Set (PDS), consisting of a set of data extents on one or more disk devices). The index in the list of Page Table Entries for a specific page is called the *Extended Page Number* (EPA#) of this page and the Extended Page Number multiplied by the pagesize (4KB) is called the *Extended Page Address* (EPA).

The Page Table describing one private area is anchored in the corresponding SCB of the space, the private area belongs to. The Page Table for the shared area is anchored in the SCB\_S (SCB of shared area).

During page handling (PMR-task), each page of virtual storage is represented by its EPA and its SCB.

Virtual address to EPA translation:

virtual address in 24-bit shared area

EPA := virt. address

virtual address in 31-bit shared area

EPA := virt. address - (size of private area)

virtual address in private area

EPA := virt. address - (size of 24-bit shared area)

The concept of real partitions is separately implemented. There is a real address space with own space control block (SCB\_R), segment table and page tables. In opposition to the virtual address spaces there is no PTAS and no POSL.

### Size of an address space

During IPL (in routine \$INTVIRT), the size of an address space is determined and used for all address spaces (COMREG.EOCADR contains the value). The minimal address space size is  $\text{MIN}(32\text{M}, \text{VSIZE})$  and the maximal address space size is  $\text{MIN}(\text{VSIZE}, 2\text{G})$ , both values rounded up to the next multiple of the segment size.

**Note:** In a non-PDS system VSIZE is calculated out of the real storage usable by page management and the specified VIO size.

The actual size of an address space is determined by the size of the shared areas and the user specified size of the private area (PASIZE). If the actual size would be above the allowed maximal value, PASIZE is decreased, and if the actual size

would be below the allowed minimal value, PAsize is increased, to fit into the address space limits.

In case the resulting PAsize is below the allowed minimal value (1M if VSIZE is smaller than 256M and 6M if VSIZE is 256M or larger) a message is given to the user.

## Handling of Data Spaces

Data spaces are handled in page management the same way as address spaces. A data space is represented by a SCB too, therefore a page fault in a data space is represented by an EPA and its corresponding SCB, the same way as a page fault in an address space (only the way to get the SCB address for a data space is different).

Besides page fault handling, page management is involved in data space support in the following areas:

- Page fault handling overlap
- Release page
- INVPAGE service
- create/delete PMR tables
- PAGESTAT service

## Distribution of Virtual Storage to Page-Data-Set

The supported virtual storage (VSIZE) is distributed in blocks of 32KB on the page data set devices. To do this, two types of tables are used.

Every block of 32KB (8 page-table entries) is associated an entry in the *Page Table Assignment String* (PTAS), containing mainly the relative record number of the 1st page of the block on the PDS. The entry is abbreviated as PTASE and contains zero if unused. The PTAS is used to get from the EPA the corresponding disk address. There exists one PTAS per address area, which is anchored in the SCB of this area.

The corresponding 32KB area of virtual storage is also called *allocation unit*.

A second table, the *Page to Disk Assignment String* (PDAS) is used to indicate whether a block of 32KB on PDS is allocated or not. The relative record number of the 1st page of the block divided by 8 is the index in this table. The allocation algorithm provides both minimal SEEK time and an uniform distribution over the extents and devices.

## Blocked Page I/O

Whenever feasible, the page I/O activities are done in blocks up to eight pages (part or total of an allocation unit) to avoid single page I/O in a multi partition and multi space environment; the aspects are both the seek overhead for each I/O and the resulting I/O contention. Blocked paging exploits the high data transfer rate capability of disks. Blocked paging is done for the following operations:

- pre-page-out,
- unconditional page-out and
- page-in (triggered by PAGEIN-macro)

## System without Page Data Set

If enough real storage is available to hold the required VSIZE and VIO, no page data set is required. The user indicates this to the system via the option NOPDS on the supervisor parameters IPL command. In this case the VSIZE parameter is not allowed and the system calculates the VSIZE (in multiples of 64K) out of the available real storage and the requested VIO space (in routine \$INTVIRT).

In a non-PDS system no page selection takes place, if a page fault occurs, there has to be at least one entry in the invalid page frame queue.

In addition, the following page-management functions result in a null operation:

FCEPGOUT - return code zero is provided in register 15.

PAGEIN - ECB, if present, is posted.

TPIN

TPOUT

## General assumptions

Due to the fact, that part of the PMR-tables can only be accessed with DAT off, part of the page management routines run in real mode. To do this the following assumptions are done:

1. Control blocks residing in an area "Virtual=Real":

BLKTBE

VTABEs (one entry per VPOOL page)

DEVCBs (device control blocks, one per PDS device)

Entry in Page-out queue (pseudo TIBs)

PGTPCB (PCB for page-out)

System PCB, PIB

SMCB

SRQPGF, SRQPGFX, SRQPGIO, SRQPFR

2. Control blocks not crossing page-boundary and being in real storage if the system is working for the corresponding partition/task.

PCB and PIB

3. Control blocks residing in 24-bit virtual:

PCB, PIB, TCB, TIB, VIOTABE and PAGETAB

---

## Control Block Allocations

With increasing address space size, the storage requirements for the page management tables increase dramatically (about 2.4 MB are required to support a private area of 1GB). Therefore the page management tables (Page Frame Table, Reentry tables, Page to Disk Assignment String, Page Table Assignment String, Page Out State List, Page Tables and Segment Tables) are allocated in one or more page manager address spaces instead of shared areas.

Page management control blocks unique in the system, like Page Frame Table, Reentry tables and Page to Disk Assignment String, are allocated in real processor storage and are addressable with DAT bit off only. They are allocated at the top of the processor storage, or below 2GB if the storage is larger than 2GB. IJBEOR (end of real storage) points to the last byte of the highest real storage frame not containing these tables. IJBEOR may be addressed by applications and therefore remains a 31-bit address. The actual physical end address of processor storage is an 64-bit value internally kept in SMCOM.SMCPEOR.

Page management control blocks not unique in the system are allocated in the private area of extra address spaces, the page management address spaces. The size of these private areas is a multiple of 64K.

## Page Frame Table, PDAS, Reentry-Rate Tables

These tables are allocated at IPL time (\$INTVIRT) from top of real storage. The sequence from the top is as follows: PDAS, RTAB, RTABX, PFT.

PDAS, RTAB, and RTABX are set on fullword boundary. The PFT will be set on page boundary.

IJBEOR is set to APFT-1.

### Page to Disk Assignment String (PDAS)

The PDAS is allocated for the total VSIZE. It requires 64KB per 2GB (one byte per 32KB)

### Reentry Rate Tables (RTAB, RTABX)

These tables are allocated adjacent to the PDAS for the total VSIZE. Access to these tables is via the record number on PDS.

### Page Frame Table (PFT)

In \$INTVIRT (during IPL) the page frame table entries belonging to the supervisor and IPL, as well as the PFTE's belonging to the area containing the PMR-tables for the 1st page manager address space and the shared address space are marked as PFIxed. All other entries are enqueued in the invalid page frame queues (IPFQ and IPFQ64). After this is done PFTE's are handled only by page-management (via INVPAGE service or page faults).

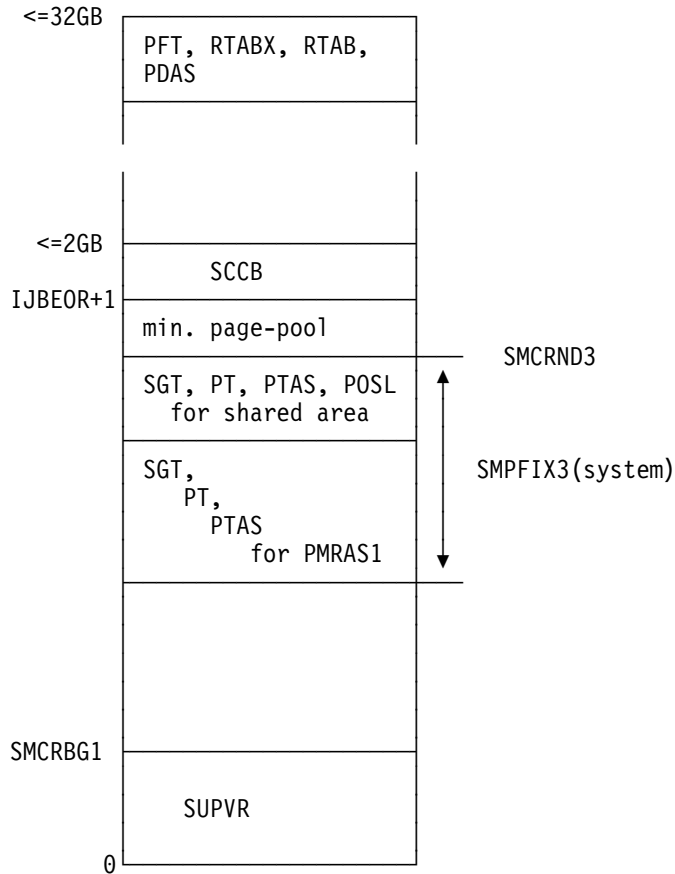


Figure 75. Real Storage Layout after \$INTVIRT

## Segment-, Page-Table, PTAS and POSL

At space creation time, these tables will be allocated and PFIxed (done in SGPPMT).

For spaces containing static partitions (not in default space), allocation is done for maximum PASIZE (Private Area Size), to ensure that the tables for one space are contiguous. Only those parts of the tables are PFIxed, which describe the actual allocated partitions.

The tables are allocated in the private area of extra address spaces (the Page Manager Address Spaces, PMRASn). Each SCB contains a pointer (SCB.SCBAPMRA) to the SCB of the PMRAS containing the tables. The 1st PMRAS (PMRAS1) is allocated at IPL time (\$INTVIRT).

The PMRAS is an address space with nearly the total private area consisting of GETVIS space. At the beginning of the private area the tables for the PMRAS itself are allocated, which don't belong to the GETVIS space.

The size of the private area of a PMRAS is calculated at IPL time to:

$$\text{MIN}((\text{VSIZE in Meg.}) * 4\text{KB}, \text{Max. private area size})$$

The PT, PTAS and POSL for private area, together with the SGT for the space are allocated in one contiguous area in PMRAS.

Allocation procedure for SGT, PT and PTAS:

- SGT on 4K boundary



- PT, PTAS and POSL each contiguous for total private area:
  - PT of private area at 1K boundary after SGT (SCB.SCBAPT contains address)
  - PTAS of private area after PT, at 1K boundary (length of page table for one segment is 1K). (SCB.SCBAPTAS contains address)
  - POSL of private area after PTAS (SCB.SCBAPOSL contains address)

**Notes:**

1. The space 0 containing the BG-partition is created at IPL time (\$INTVIRT), by issuing a ALLOCATE request for BG and running in PMRAS1
2. The shared space containing the shared area is created at IPL time (\$INTVIRT).

**Segment-tables**

At creation time of an space the segment table will be created (at 4K boundary) with invalid entries for the private area of the space (STEINV=ON). The Segment table will be allocated in the PMRAS and PFIxed. The length of the segment table for an address space is determined during IPL and a multiple of 4096 bytes (512 entries).

The length of the segment table of a data space is calculated at allocation time and a multiple 4096 bytes (512 entries).

The Segment-table is freed up at space termination time.

**Note:** The private area of a data space is the whole space.

**Page-tables**

Page tables will be allocated at 1K boundary for all usable segments.

The page-tables for the shared areas are allocated and PFIxed at IPL time (\$INTVIRT).

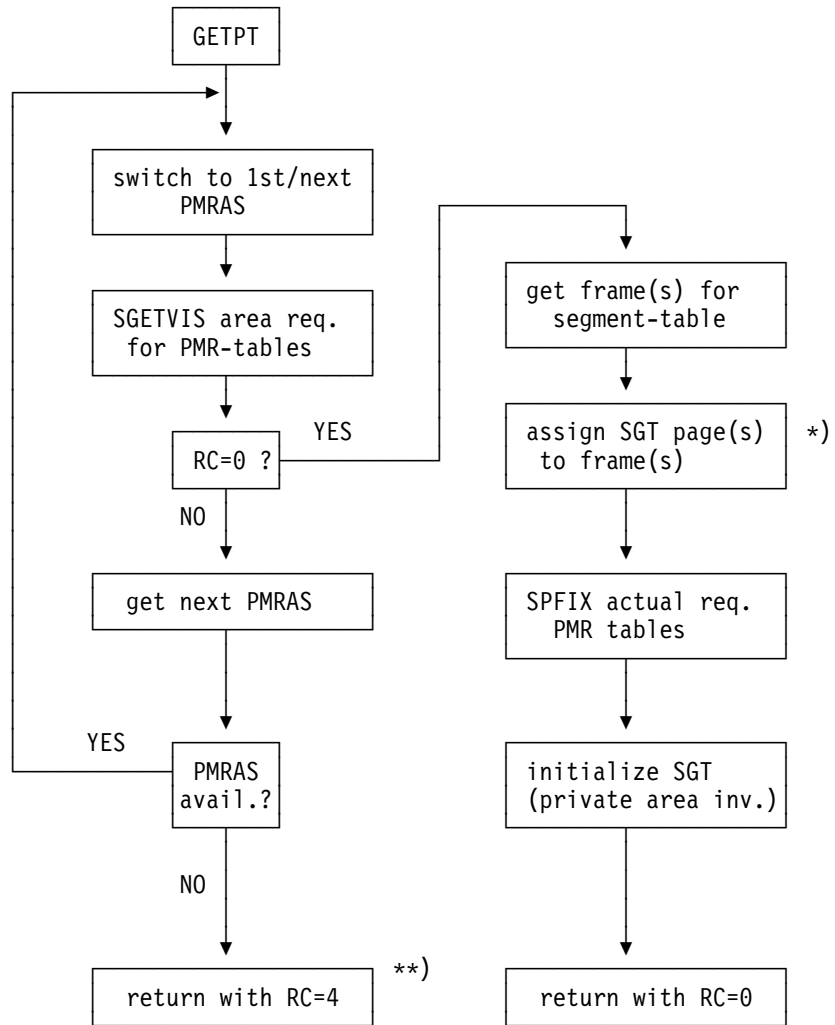
The page-tables for the private segments of an address-space are allocated in a PMRAS during creation of the space, they will be freed up at space termination time.

**Page-table Assignment String (PTAS)**

The PTAS for a segment is allocated in the same area and together with the Page-table (at 8-byte boundary).

**Page-Out State List (POSL)**

The POSL for a segment is allocated in the same area and together with the Page-table (at 8-byte boundary).



\*) Can't be done via SPFIX, since Segment table must be contiguous even in real storage and may require more than 4KB.

\*\*\*) Results in an allocation request for a new PMRAS.

Figure 76. Allocation of PMR Tables

Since the PMR tables for a PMRAS are allocated in the PMRAS itself, SGETVIS and SPFIX can't be used in this case. The frames for the PMR tables have to be obtained "manually" (using GETREAL) and the pages containing the tables have to be assigned to the frames while running in real mode.

## Handling of Space R (Real Address Space)

The segment and page tables for space R will be allocated and PFIxed in a PMRAS during 1st ALLOC of a real partition. If there are no more real partitions allocated, the segment table and page tables are freed up again.

## PMRAS space layout

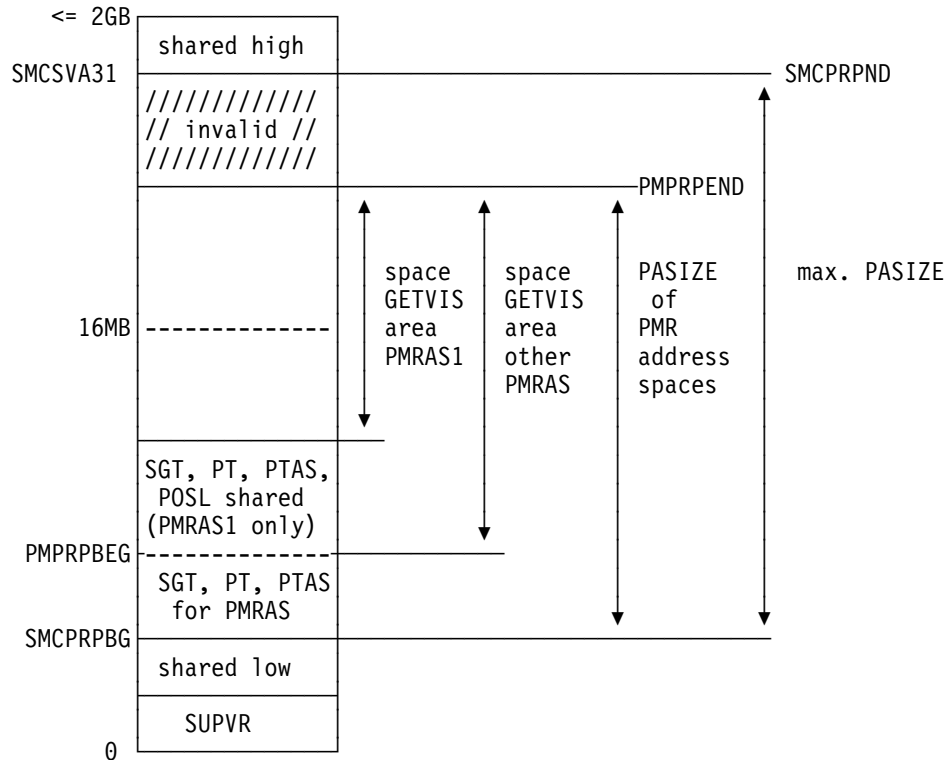


Figure 77. Layout of Page Manager Address Space (PMRAS)

## Data Structures of Page Management

The Segment Table (ST) with its Segment Table Entries (STE-s) describes an address space or a data space. Each STE represents a 1MB address range and points to the related Page Table (PT) if the STE is valid.

### Segment Table

The virtual storage is organized in logical spaces up to 2GB. There are:

- address spaces
- data spaces

Each logical space (virtual memory) is identified by its segment table. At IPL time the complete segment table is generated for the shared address space and the first private address space. This table contains one entry for each segment of virtual storage (segment size: 1MB).

The segment tables for the other address spaces are allocated and initialized whenever the first partition of this address space is activated and the related Space Control Blocks (SCB) are built. The SCB provides a pointer to the associated segment table origin.

The segment table entry is defined by the z/Architecture (see *Principles of Operation*, SA22-7832), as shown in the following Figure 78 on page 200:

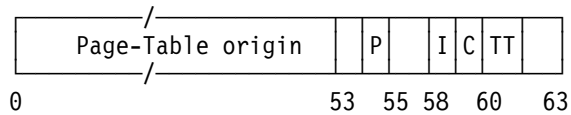


Figure 78. Segment Table Entry z-Mode

page table origin:

with 11 zeroes appended on the right, forms the address of the page table for this segment. For 2GB address spaces the address fits in 31-bits.

P: Page-Protection Bit

I: invalid segment bit ( = 0 - the segment is valid)  
( = 1 - the segment is invalid)

C: common segment bit

TT: identifies the level of the table.  
00 means segment table.  
(Non-zero identifies region tables, not applicable for VSE)

## Page Table and Page Frame Table Entries

The unit of virtual storage is the page of the size of 4KB. It is represented by the associated PTE which describes the state of a page.

A page is addressable, if it is located in a page frame; it is disconnected, if it is not in a page frame; it is connected if it is located in a frame but not addressable (connected state only used for page management purposes, during page I/O ) and it is invalid, if the page does not exist (doesn't belong to any address space).

The PTE is given by the z/Architecture (see *Principles of Operation, SA22-7832*) if the invalid bit is off. If the invalid bit is on the PTE is interpreted by the VSE/AF software as shown in Figure 80 on page 201.

Bits	Label	Description
0-63	PTE	Page addressable
0-51	PTEFRA64	Page frame number
32-51	PTRFRA	31-bit page frame number
52		X'08 Architected = 0
53	PTEIBIT	X'04' Invalid bit = 0
54		X'02' Protect.bit = 0
55		X'01' Architected = 0
56-62		unassigned
63	PTEPDS	X'01' copy on page data set

Figure 79. Page Table Entry (PTE) for Addressable Page

Bits	Label	Description
0-63	PTE	Page not addressable
0-39		Filler
40-43	PTEKEY	Storage key of page
44		X'08' fetch protection bit
45-47		reserved
48	PTEERR	X'80' Erroneous page: PTEERR=PTEIBIT=1
49	PTENASS	X'40' Page not assigned: PTENASS=PTEIBIT=1
50	PTEINVAD	X'20' Invalid state: PTEINVAD=PTEIBIT=1
51	PTEPGCO	X'10' Connected state: PTEPGCO=PTEIBIT=1
52		reserved
53	PTEIBIT	X'04' Invalid bit = 1
54-62		reserved
63	PTEPDS	X'01' Valid copy on PDS = 1 no copy on PDS = 0

Figure 80. Page Table Entry (PTE) for not Addressable Page

The page has addressable state: PTEIBIT = 0.

That means the page is currently in real storage and the frame is given by the PTEFRA value.

$$\text{PTEFRA} = \text{frame-address} * 2^{**}(-12)$$

The page has invalid state: PTEIBIT = PTEINVAD = 1 and PTEPGCO = 0.

That means the page is not in the address range of the memory, for example, a reference to the real partition if the virtual partition is active.

The page has connected state: PTEIBIT = PTEPGCO = 1 AND PTEINVAD = 0.

That means page I/O is running for this page.

The page has disconnected state: PTEIBIT = 1 and PTEINVAD=PTEPGCO = 0.

That means the page is not in real storage.

The page is not assigned: PTEIBIT = PTENASS = 1 AND PTEPGCO = 0.

That means no slot on the Page Data Set device is assigned to this page.

A data invariant is given as: PTEINVAD=PTEPGCO=1 not possible.

## Page Table Initialization

1. During IPL, page table entries for shared space and for space 0 are initialized as follows:

- All page table entries belonging to the supervisor area (nucleus and transient areas):

PTEFRA Number of the corresponding page frame

PTEIBIT=PTEPDS = 0

All other bits = 0

that means, the pages are addressable.

- Page table entries belonging to VIRTUAL BG partition:

```

PTEKEY Storage key of corresponding partition
PTEINVAD = 0
PTENASS = 0
PTEPGCO = 0
PTEIBIT = 1
All other bits = 0

```

that means, the pages are disconnected.

- Page table entries belonging to SVA-24 and SVA-31:

```

PTEKEY Storage key of SVA
PTEINVAD = 0
PTENASS = 0
PTEPGCO = 0
PTEIBIT = 1
All other bits = 0

```

that means, the pages are disconnected.

- All remaining page table entries:

```

PTEKEY = 0
PTEINVAD = 1
PTENASS = 1
PTEPGCO = 0
PTEIBIT = 1
All other bits = 0

```

that means, the pages are invalid and not assigned.

The storage key is part of frame and must be saved in the PTE whenever the page is disconnected.

## Page Frame Table (PFT)

The real storage is subdivided into page frames of the size of 4KB. Each frame is uniquely associated to an entry in the PFT describing the status of the frame. This entry is abbreviated as PFTE.

The page selection queue (PSQ) contains all PFTEs of frames occupied by pages and usable for page replacement (essentially pages which are not FIXed). The number of PFTEs in PSQ is given by len(PSQ).

The invalid page frame queue (IPFQ) contains all free PFTEs. The number of PFTEs in IPFQ is given by len(IPFQ).

Page frames below 2GB are organized in the IPFQ and PSQ as described above. The page frames above 2GB are managed in two separate PFTE queues, the IPFQ64 and the PSQ64. They have the same format queuing free 64-bit page frames and 64-bit page frames available for replacement. This is done because 31-bit page frames and 64-bit page frames cannot be used transparently. There are restrictions for the use of 64-bit page frames: They cannot be PFIxed or TFIxed. The separation helps to keep searches for fixable page frames short, and to make explicit selection of 31-bit page frames and 64-bit page frames easier and faster.

Since only 31-bit page frames can be used for loading programs or fixing data areas, 64-bit frames have a relief function for the area below 2GB. Active 31-bit page frames may be replaced by 64-bit frames before being reused.

The PFT is built at IPL time and contains one 32-byte entry for each real storage block of 4KB. Field APFT (in SUPAVT) contains the begin address of the table.

Figure 81 on page 203 shows the layout of a page frame table entry (PFTE).

PFTE FLG	Extended page # PFTEEPA#	S370 FLG	PFTE FLG3	PFTE- PIK	PFIX Count	TFIX Count	PFTE- DVCB	PFTE- TIB	PFTE- SCB	PFTE- FPTR	PFTE- BPTR	
0	1	4	5	6	8	10	12	16	20	48	56	63

Figure 81. Page Frame Table Entry (PFTE)

Byte(s)	Label	Description
0	PFTEFLG	PFTE flag
		X'80' Reserved
	POEBIT	40 The PFTE is enqueued for page-out.
		20 Reserved
	POABIT	10 I/O for a page-out has been started for this PFTE.
	PCBIT	08 The page which belongs to the page frame has connected state. Either a page-in or an unconditional page-out request is in progress.
	POSYSBIT	04 A page-out request is in a system queue.
	PIOERR	02 I/O error on frame
	PFTEQBIT	01 PFTE in enqueued (only used with DEBUG)
1 - 3	PFTEEPA#	If a page belongs to the page frame, these bytes contain the Extended Page Number (index in page table pointed to by PFTESCB.SCBAPT) If a block of VIO storage belongs to the frame, these bytes contain the block number.
4	S370FLG	370 mode flag
	NFRP	X'80' Frame is reserved for PFIx or GETREAL. If the TFIx counter is zero, the page must no more be TFIxed. If the frame is in IPFQ or in PSQ, NPSQE is decreased by one.
	NFVP	40 Page belonging to this frame is requested by PFIx. The frame is not in the PSQ. The PFIx request cannot be satisfied immediately.
	DRAP	20 The address space belonging to the PFTE is failing storage.
	PFTEBLK	10 Only block of VIO-storage connected to frame
	PNRINV	08 Page frame is unused. The content of the PFTE is invalid, except NFRP, PFTERES and DRAP bits in S370FLG.
	PFTERREAL	04 Frame is used by real partition.
	PFTERES	02 Frame is reserved, don't PFIx 01 Reserved
5	PFTEFLG3	3rd PFTE flag
	POSLISON	80 marked for page-out 40..20 reserved
	CURSELCT	10 current selection 08 reserved
	PFTEXCH	04 A(PFTE)/chaining inconsistent (31/64 Q)
	USPGBIT	02 used for internal statistics only
	RCLBIT	01 used for internal statistics only
6 - 7	PFTTEPIK	PIK of waiting partition, requesting PFIx. The page frame of the page to be PFIxed does not belong to the corresponding real area.
8 - 9	PFIx	Indicates how often the page is PFIxed.
10 - 11	TFIx	Indicates how often the page is TFIxed.
12 - 15	PFTEDVCB	Addr. of DEVCB, waiting for this frame.
16 - 19	PFTETIB	Addr. of page out TIB
20 - 23	PFTESCB	Addr. of SCB the EPA belongs to
24 - 47		Reserved
48 - 55	PFTFPTR	Pointer to the next PFTE.
56 - 63	PFTBPTR	Pointer to the preceding PFTE.

Figure 82. PFT Entry Byte Description

**Note:** The pointers in bytes 24-31 are only valid if the PFTE is in the PSQ or IPFQ.



## Status of a Page Frame Table Entry (PFTE)

1. If a PFTE is not assigned to a page:
  - If no block of VIO-storage is connected to the frame, the PFTE is enqueued to the Invalid Page Frame Queue (IPFQ), the PNRINV bit is set, and the NFRP, PFTERES bits may be set, and the remaining contents of the PFTE is undefined.  
If only a block of VIO-storage is connected to the frame, the PFTE is enqueued to PSQ, the TFIX and PFIX counters are zero and PFTEBLK bit is set.
2. If a PFTE is assigned to a connected page:
  - The PFTE is neither enqueued to the PSQ nor to the IPFQ. The contents of the PFTE is valid. The PCBIT is set, PFTEEPA# indicates a connected page, PFTESCB contains the SCB of the corresponding space and the PFIX and TFIX counters are zero.
3. If a PFTE is assigned to an addressable page, the contents of the PFTE is valid:
  - If the NFVP bit is set, the PFTE is neither enqueued to the PSQ, nor to the IPFQ. If the NFVP bit is reset, and the PFIX and the TFIX counter are zero, the PFTE is enqueued to the PSQ. If the NFVP bit is reset, and the PFIX or the TFIX counter is not zero, the PFTE is neither enqueued to the PSQ nor to the IPFQ.

## NPSQE

NPSQE represents the actual value of 31-bit page frames available for replacement. That means:

$$\text{NPSQE} = \text{len (PSQ)} + \text{len (IPFQ)}$$

In order to prevent excessive page fixing and thus guarantee program execution under all conditions, the number of available page frames must not be lower than a specific limit MINPSQE. That means there is data invariant:

$$\text{NPSQE} \geq \text{MINPSQE}$$

The reservation of MINPSQE, or if it is a request from the Fetch routine, of MINPSQE-2 page frames for page replacement ensures that a page fault can always be handled by the PMR task.

If the PFIXPGE or GETREAL routine is executed, the counter NPSQE does not reflect all the time the actual number of PFT entries in the PSQ. The actual number of entries in the PSQ can be greater than the number indicated in NPSQE. Those additional entries are reserved by the PFIXPGE or GETREAL routine and cannot be used for other requests. Please see sections Page Frame Table and Selection Pool Queues.

NPSQE monitors only the availability of 31-bit page frames. 64-bit page frames cannot be fixed and therefore are not taken into account.

## Page Table Assignment String (PTAS)

Every block of 8 page table entries, describing a contiguous address range of 32KB is associated to an entry in the *Page Table Assignment String* (PTAS). It indicates whether the related block is already in use or not. The entry is abbreviated as PTASE and contains zero if unused. Field SCBAPTAS in SCB contains the begin address of the table describing the area belonging to the SCB.

Bytes	Label	Description
0 - 7	PTASE	Entry length 8 bytes
0 - 1		Reserved
2	PTASFLG1	Flag byte 1
	PTASUSED	X'80' entry in use 40..01 Reserved
3		Reserved
4 - 7	PTASRECN	Record# on PDS of 1st page of this block

Figure 83. Page Table Assignment String Entry (PTASE)

## Page to Disk Assignment String (PDAS)

Every block of 8 pages on the page data set (PDS) describing a contiguous area of 32 KB (the so called allocation unit) is associated to an one byte entry in the *Page to Disk Assignment String* (PDAS). It indicates whether the related block is already in use or not. The entry is abbreviated as PDASE and contains zero if unused, and X'FF' if it is used. Field APDAS in PMCOM contains the begin address of the table.

**Note:** The PDAS is only addressable with DAT off.

## Page State Lists (PSLs)

The *Page State List* (PSL) describes the states of the pages of an allocation unit.

The state descriptor of the PSL is implemented as a bitstring, indicating whether the condition is satisfied or not.

Bytes	Label	Description
0 - 15	PSL	Header of PSL
0 - 3	PSLASCB	Address of SCB the PSL belongs to (PISL only)
4 - 7	PSLABEGP	EPA number of 1st page.. ...of related page set
8 - 11	PSLAENDP	EPA number of last page.
12 - 15	PSLSTLEN	length of PSLSTATE
16 -	PSLSTATE	array of Boolean values ... describing the ... states of page set

Figure 84. Page State List (PSL)

Currently there are two PSLs:

- POSL (Page-Out State List)

- PISL (Page-In State List)

The POSL is provided by the page selection algorithm and is addressed via SCBAPOSL in the related SCBs.

The PISL is set up by the page-in SVC routine and is located in the page manager data area.

The space requirements of a PSL is 1KB per 32MB address space (or 32KB per 1GB).

## Pre-Page-Out Control Table (PREPGOTB)

The Pre-Page-Out Control Table (PREPGOTB) provides information to manage pre-page-out processing via the current POSL.pslstate. PREPGOTB consists of 256 PREPGENT entries.

Bytes	Label	Description
0 - 7	PREPGENT	PREPGENT entry
0 - 1	PREPGBLK	nbr of contiguous blocks relative EPA nbr of 1st ...page in allocation unit
2	PREPGPTE	
3	PREPGVAL	blocking value mask
4 - 7	PREPGOFF	offset of first page in ... TIBAPFT

Figure 85. PREPGENT Entry

## Page-In Control Table (PAGEINTB)

The Page-In Control Table (PAGEINTB) provides information to manage blocked page-in processing via the current PISL.pslstate. PAGEINTB consists of 256 PGINENT entries.

Bytes	Label	Description
0 - 7	PGINENT	PGINENT entry
0 - 1	PGINBLK	nbr of contiguous blocks number of required frames ...to satisfy page-in
2	PGINPFR	
3	PGINVAL	blocking value mask
4 - 7	PREPGOFF	offset of first page in ... TIBAPFT

Figure 86. PGINENT Entry

## Storage Management Control Block (SMCB)

The SMCB - being part of the partition control block (PCB) - contains the necessary control information for the storage allocation. The page management is concerned by:

- SMAXPFIX     partition/SVA PFIX limit in pages (24-bit)
- SMPFIX        actual PFIX count (24-bit)
- SMAXPFIX3    partition/SVA PFIX limit in pages (31-bit)
- SMPFIX3       actual PFIX count (31-bit)

Moreover, the virtual and real partition boundaries are considered by the page management.

## Page Data Set Table

Page Management uses the Page Data Set Table (DPDTAB) to calculate the correct address for a given page on the Page Data Set, if a read or write operation is necessary. Bytes 224-227 (X'E0'-X'E3') of the System Communication Region (SYSCOM.IJBDPDTB) contain the address of the DPDTAB. The DPDTAB consists of a header and 15 extent definitions. Label DPDTAB identifies the first byte of the table. The table has the following layout:

Dec	Hex	Label	Description
0-15	0- F	DPDADR	Header
0- 1	0- 1	DPDEXT#	Number of possible extents
2- 3	2- 3	DPDAEXT#	Number of actual extents
4- 7	4- 7		Reserved
8-11	8- B	DPDLLCON	Address of load leveling constants
12-13	C- D		Reserved
14-15	E- F	DPDLEN	Length of one DPDENTR

Figure 87. Page Data Set Table Header

Dec	Hex	Label	Description
0-39	0-27	DPDENTR	Extent definition
0-1	0-1	DPDUNT	CUU of PDS device
2	2	DPDDEV	Device type:FBA, CKD, RPS
3	3	DPDDEV	Device code (DTF)
4-5	4-5	DPDREC#	CKD: # records/track
4-5	4-5	DPDBLKLG	FBA: block length
6-7	6-7	DPDTRCK#	CKD: # tracks/cylinder
6-7	6-7	DPDBLKPG	FBA: # blocks/page
8-11	8-B	DPDRTLL	CKD: track# of lower extent limit
8-11	8-B	DPDBLKLL	FBA: block# of lower extent limit
12-15	C-F	DPDTRCKU	CKD: # of used tracks
12-15	C-F	DPDBLKU	FBA: # of used blocks
16-17	10-11	DPDPUB	PUB index
18-23	12-17	DPDVOLID	Volume id of PDS
24-27	18-1B	DPDPGUL	Page # of upper limit
28-31	1C-1F	DPDXNTLL	ECKD, ext. lower limit
32-35	20-23	DPDXNTUL	ECKD, ext. upper limit
36-39	24-27	DPDDEV	Addr. of DEV
40-41	28-29	DPDFCP	SCSI: CUU of FCP
42	2A	DPDFFLG	SCSI: Dump flags
		DPDDOP	80 - SCSI extent open
		DPDXERR	40 - error on SCSI extent
43	2B		Reserved
44-51	2C-33	DPDWWPN	SCSI: Port number
52-59	34-3B	DPDLUN	SCSI: LUN number

Figure 88. Page Data Set Extent Definition

## Device Control Block (DEVCB)

Every PDS device is described by its associated Device Control Block (DEVCB).

Bytes		Label	Description
Dec	Hex		
0	0	DEVCB	Device control block
0- 3	0- 3	DEVCBNXT	Addr. of next DEVCB if any, addr. of first DEVCB in chain for last DEVCB
4	4	DEVSTAT	Status byte
		DEVSTRT	X'80' I/O request started
		DEVEMPTY	X'40' no I/O request enqueued
		DEVPGWO	X'20' request waits for unconditional page out
5	5	DEVCBTYP	Device type: FBA,CKD,RPS,ECKD
6	6	DEVEXT#	Number of extents on device
7	7	DEVCP SL	PSLSTATE of active CCW-s (blocked I/O)
8- 11	8- B	DEVACT	Address of IORE (TIB)
12- 15	C- F	DEVDPD	Addr. of 1st DPD entry for device
16- 19	10- 13	DEVRELO	Relocation for 1st DPD entry on device
20- 23	14- 17	DEVAPDAS	Addr. of 1st PDASE for device
24- 27	18- 1B	DEVPDASA	Highest offset of PDASE already occup.
27- 31	1C- 1F	DEVPDASB	Number of PDASEs to be scanned
32- 35	20- 23	DEVPCB	Address of related Class PCB
36- 39	24- 27	DEVCBMSK	Device mask for dispatching (turbo)
40- 41	28- 29	DEVERIN	number of failing blocked page-in
42- 43	2A- 2B	DEVEROUT	number of failing blocked page-out
44- 47	2C- 2F	APFPSS	Address of PFPSS for device
48- 63	30- 3F	DEVCCB	CCB for device
64- 71	40- 47	DEVCCW	CCW program area
72-471	48-1D7		Device specific information
472-475	1D8-1DB	PFRQBEG	Begin addr. of system page fault queue
476-479	1DC-1DF	PFRQEND	End addr. of system page fault queue
480-655	1E0-28F		Partition queue headers in the sequence BG, FB, ... , F1,Classes.. length = (12+NCLASS)*2*4 NCLASS = number of dynamic classes
656-659	290-293	PORQBEG	Begin address of page-out queue
560-663	294-297	PORQEND	End address of page-out queue

Figure 89. Device Control Block (DEVCB)

## Page I/O Request Element (IORE)

The IORE is part of Task Information Block (TIB). The following fields are relevant for page management.

Dec	Hex	Label	Description
0	0	TIBADR	Task information block
0-3	0-3	TIBCHAIN	
4-7	4-7	TIBSTATE	Bound state information page-in: ext. page addr. of page-fault blocked page-in: address of PGIN-TIB unc. page-out: addr. of PFTE triggering the page-out pre page-out: binary zeros
8-11	8- B	TIBPFAPP	Address of PHO appendage (PHO-TIB)
8-11	8- B	TIBVIOTB	Address of VIOTAB entry (VIO-TIB)
12	C	PGQTYP	Request type
		PGPMRSP X'80'	page-out for page-manager address space
		PGNCNT X'40'	Page-in: counting done
		PGIN X'20'	Page-In req. by PGIN task
		PGO X'10'	Page-out request
		PGIOERR X'80'	Page I/O error occurred
		PGDELO X'04'	page-out: deactivate request
		PGVIORQ X'02'	VIO request
		PGBLK X'01'	Blocked Page I/O request
13	D	TIBFLAG1	Flag byte
		PHOIND X'80'	indicates PHO/VIO TIB
		PHOACT X'40'	PHOIND=0: PHO initialized for this task
		PHOREQ X'40'	PHOIND=1: Req. enqueued
		VIOREQ X'20'	PHOIND=VIOREQ=1 indicates VIO TIB
14	E	TIBFLAG4	Flag byte
		TIBPFDSP X'40'	PHO for page-fault in data spaces active
16-19	10-13	PGINF	Information for page I/O handling page-in: addr. of PFTE blocked page-in: binary zeros page-out: binary zeros / ... / address of waiting DEVCB
20	14	PGOEQPSL	pslstate at ENQUEUE time
21	15	PGOIOPSL	blocked part of pslstate at SVC-15-time
22	16	PGOCYPSL	blocked part of pslstate if end-of-cylinder has been detected
24	18	TIBPCB	Pointer to PCB the task belongs to .... further TIB
.	.		
.	.		
28-31	1C-1F	TIBAALU	page-out: addr. of allocation unit in current PSL
32-35	20-23	TIBPFSCB	SCB the TIBSTATE belongs to
36-39	24-27	TIBPFARA	Addr. of PHO interface area
36-39	24-27	TIBERPFT	offset of failing CCW (blocked I/O)
40-71	28-47	TIBAPFT	array of addr to PFTE-s (blocked I/O)
.	.		

Figure 90. Page I/O Request Element (IORE)

# Relationships between Control Blocks

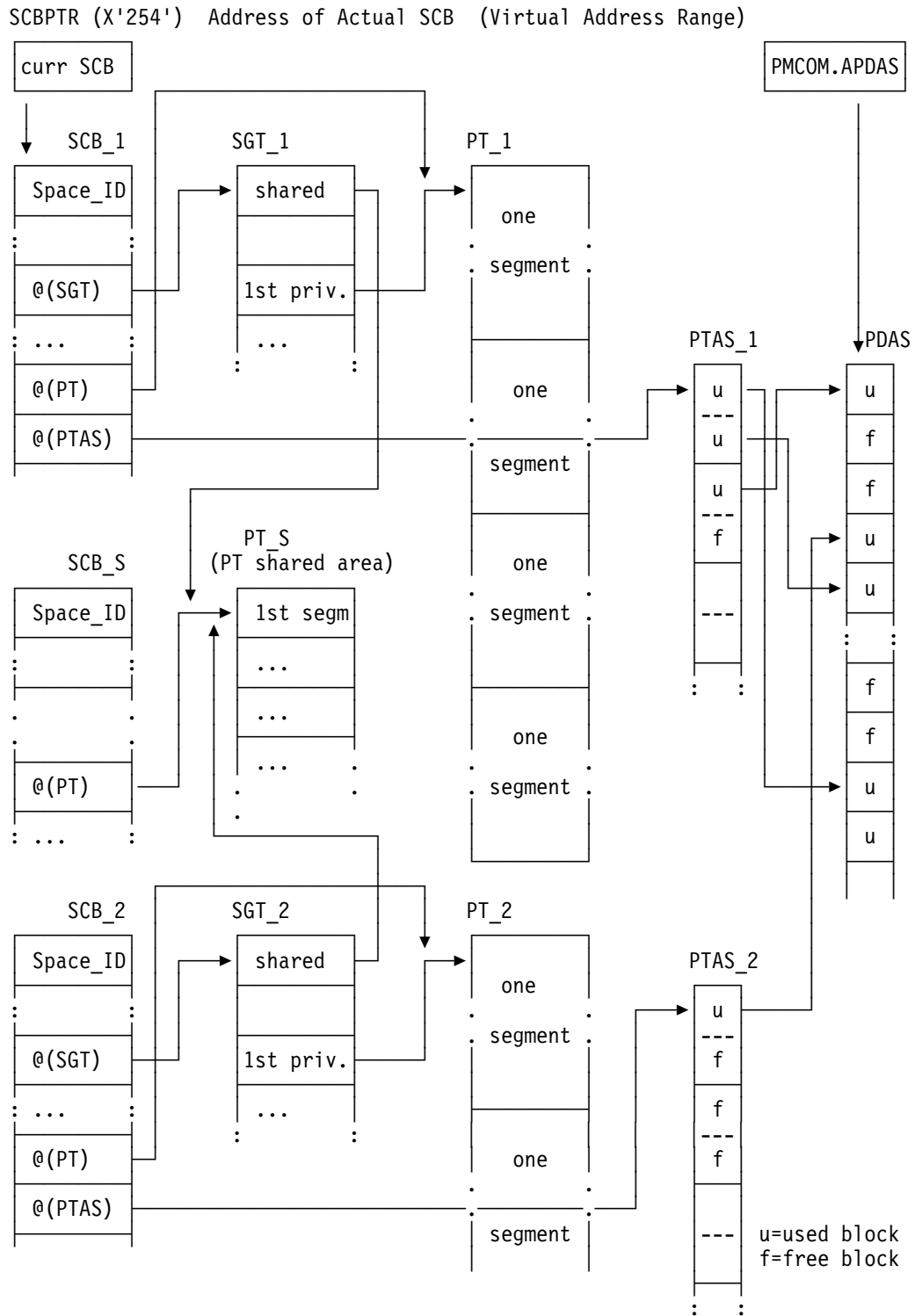


Figure 91. Relations between SCB, Segment Table, Page Table, PTAS and PDAS



$L'PDAS$   $L'PDAS = \text{Length of PDAS}$

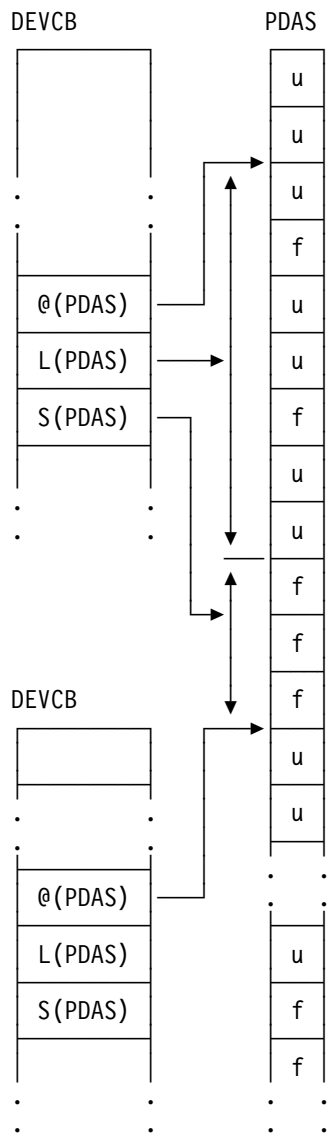


Figure 92. Relations between DEVCB and PDAS

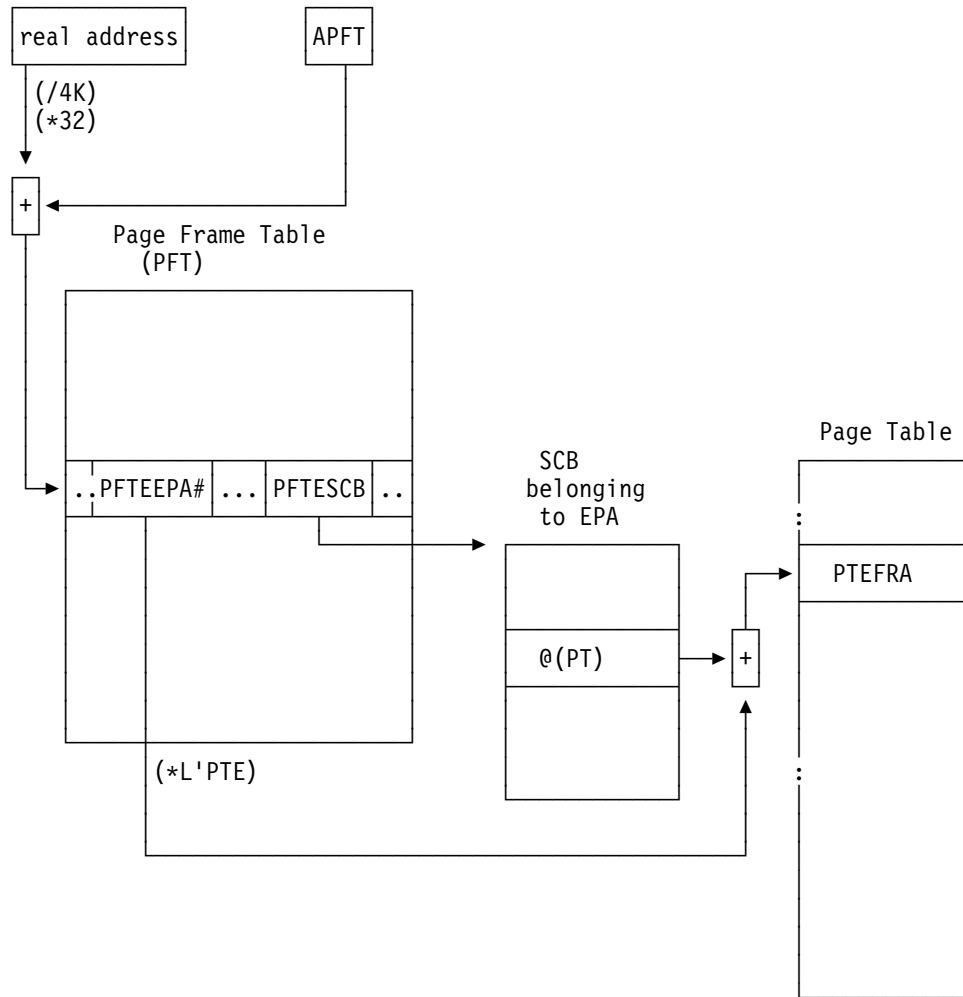


Figure 93. Relations between PFTE, SCB and PT

## Page Faults and Page Frame Selection

### Page Fault Processing

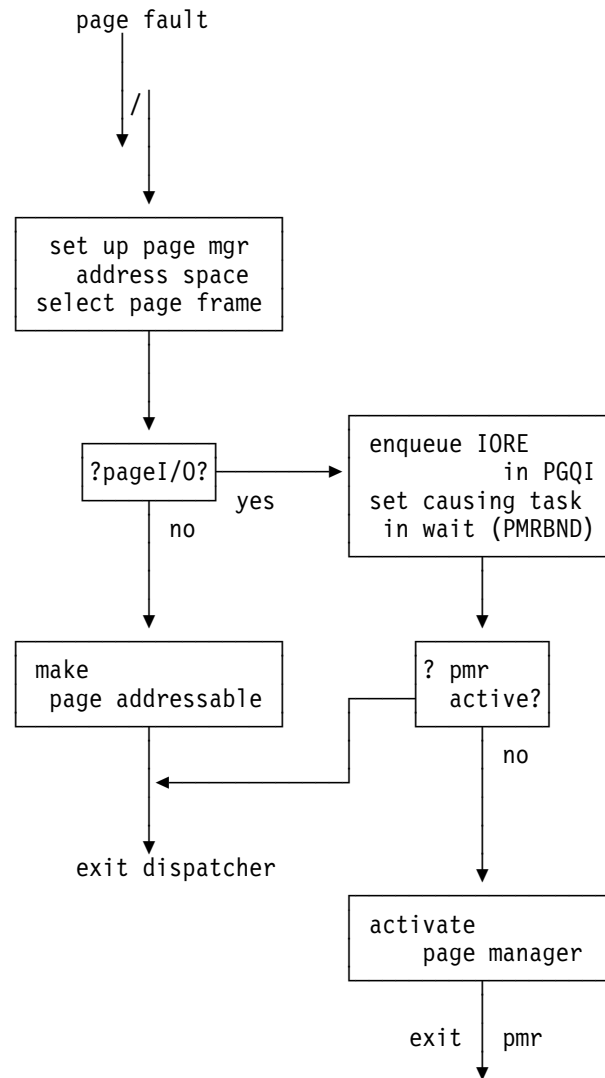


Figure 94. Page Fault Processing

Page fault handling is done under the causing task as long as possible. But whenever page I/O processing is needed the page manager task is concerned. Page fault handling is done synchronously and the task causing the page fault is set into wait (PMRBND condition). After successful completion of the page I/O the page is made addressable and the task causing the page fault is posted. Page I/O for pre-page-out and page-in SVC is done asynchronously.

## Selection Pool

The selection pool consists of all page frames which can be selected by the Page Management routines for paging. The selection pool contains all those pages which do not belong to the supervisor or to active real partitions and which are not fixed in some way (either by TFIX or PFIX).

## Selection Pool Queues

The PFTEs that are not fixed (TFIX and PFIX counter zero) and have a page assigned are queued in the Page Selection Queues (PSQ/PSQ64). The PFTEs that have no page assigned are queued in the Invalid Page Frame Queues (IPFQ/IPFQ64). One of each queues is for page frames below 2GB and the other (...64) for page frames above 2GB.

Each queue has a queue header, which is 32 bytes long. Bytes 24 through 27 point to the first queue entry and bytes 28 through 31 to the last queue entry. How the selection pool page frame entries are queued is explained in the following section and in the section on the page frame selection.

## Selection Algorithm

**Note:** The reference (R) bit and the change (C) bit are located in the page frame. Whenever they are mentioned in this paragraph they refer to the page frame or the page belonging to the entry presently handled.

Unused page frames are available if the IPFQ64 or IPFQ are not empty and they are selected by using the first one in the queue. First the the page frames from IPFQ64 are taken, then the page frames from IPFQ.

To ensure that pages newly paged-in are not paged-out immediately, they are enqueued at the end of the PSQ, the R-bit=OFF and C-bit=OFF.

To overlap the page-in and page-out functions and to avoid the necessity of executing a page-out immediately before a page-in, a pre-page-out is implemented. It ensures that a minimum number of page frames is available (that means, the page belonging to a frame has its R-bit=OFF and C-bit=OFF). The pre-page-out is only active if IPFQ is empty.

The two functions of the page selection algorithm are:

- To select a page to be replaced.
- To ensure that a pre-page-out is executed if necessary.

To achieve this, the PSQ is scanned and the state of the R- and C-bit is checked.

**Note:** The following algorithm applies to the PSQ only. Page selection from the PSQ64 has not been implemented. The PSQ is scanned till a minimum number of entries with the R-bit=OFF has been found. If all these entries have their C-bit=OFF, too, the selection is finished, and the first PFTE found in the PSQ with the R-bit=OFF and the C-bit=OFF is used for replacement.

The first PFTE in the PSQ with the R-bit=OFF and the C-bit=OFF is used for replacement. If no such entry is found, the first PFTE with the R-bit=OFF is used.

For each PFTE found during the scan and with the R-bit=OFF and the C-bit=ON, routine ENQUOBLK is called to mark the frame for pre-page-out and schedule pre-

page-out if necessary. Each PFTE found with the R-bit=ON is enqueued at the end of the PSQ with the R-bit=OFF.

## Decision Tables Used for Page Selection

Conditions:										
R-bit	1	1	1	0	0	0	0	0	0	0
C-bit	1	0	0	1	1	1	0	0	0	0
SELCTFLG =NEWSTRT	-	-	-	1	0	-	1	0	1	0
SELCTFLG =REPLFND	-	-	-	-	-	-	1	0	0	0
POABIT	-	1	0	0	0	1	-	0	0	1
Actions:										
SELCTFLG =X(NEWSTRT)					x			x		x
SELCTFLG =REPLFND								x	x	
PSQRPTR					x			x	x	x
PSQ= t1(PSQ)+hd(PSQ)	x		x							
PFTEPSL (reset)	x		x							
ENQUOBLK				x	x					
incr CNT1		x		x	x		x	x	x	x
reset R-bit	x	x	x	x	x	x	x	x	x	x
next PFTE	x	x	x	x	x	x	x	x	x	x

*Figure 95. Decision Table for Inspection of a Frame*

Conditions:			
PSQRPTR	n	y	y
SELECTFLG =REPLFND	-	0	1
Actions:			
PMRIOWT	x		
PAGDISCA			x
ENQUOUNC		x	

Figure 96. Decision Table for Page Frame Replacement

## Rearranging of Page Selection Queues

Page frames below 2GB are organised in the PSQ and IPFQ.

1. The PFTE of a page frame below 2 GB is dequeued from the PSQ:
  - If a TFIX or PFIX is requested for a page assigned to a page frame.
  - If the page assigned to a page frame has to be disconnected next (SELECTPG, INVPAGE, RELPAG).
  - If GETREAL is requested for the page frame.
2. The PFTE of a page frame below 2GB is enqueued to the PSQ:
  - If a page has been TFREEed and is otherwise not fixed, and the NFVP bit is reset (if TFREE is from Fetch, the PFTE is enqueued at the beginning of the PSQ; if it is not from FETCH, the PFTE is enqueued at the end of the PSQ).
  - If a page has been PFREEed and is otherwise not fixed (the PFTE is enqueued at the end of the PSQ).
  - If a page-in has been completed (the PFTE of the page frame assigned to the page is enqueued at the end of the PSQ).
3. The PFTE of a page frame below 2 GB is moved within the PSQ:
  - If during page selection a page is found with the R-bit on (PFTE is enqueued at the end of the PSQ).
  - If a PAGEIN request is for a page that is already in storage (the PFTE is enqueued at the end of the PSQ).
  - If a FCEPGOUT request is for a page that is in storage (the PFTE is enqueued at the beginning of the PSQ).
4. The PFTE of a page frame below 2GB is enqueued at the beginning of the IPFQ:
  - If no page is assigned to the page frame (after disconnect, INVPAGE RELPAG, FREEREAL).
5. The PFTE of a page frame below 2GB is dequeued from the IPFQ:

- If an unfixed page frame is needed and the IPFQ is not empty during page selection, and in case of TFIX, PFIX and GETREAL to exchange page frames.

Page frames above 2GB are organised in the PSQ64 and IPFQ64.

1. The PFTE of a page frame above 2GB is dequeued from the PSQ64:
  - If the page assigned to a page frame has to be disconnected next (INVPAGE, RELPAG).
2. The PFTE of a page frame above 2GB is enqueued to the PSQ64:
  - If a page-in has been completed (the PFTE of the page frame assigned to the page is enqueued at the end of the PSQ64).
3. The PFTE of a page frame above 2 GB is moved within the PSQ64:
  - If a PAGEIN request is for a page that is already in storage (the PFTE is enqueued at the end of the PSQ64).
  - If a FCEPGOUT request is for a page that is in storage (the PFTE is enqueued at the beginning of the PSQ64).
4. The PFTE of a page frame above 2GB is enqueued at the beginning of the IPFQ64:
  - If no page is assigned to the page frame (after disconnect, INVPAGE RELPAG, FREEREAL).
5. The PFTE of a page frame above 2GB is dequeued from the IPFQ64:
  - If an unfixed page frame is needed and the IPFQ64 is not empty during page selection, and in case of page-out for a page frame below 2GB to exchange page frames.

### Selecting Page Frames above 2GB

Page faults that are reported via interrupt usually occur in applications, that are not interested in the real address of the page. Therefore preferably page frames above 2GB (64-bit page frames) are assigned to the page.

Internal page faults are queued by system functions like Fetch, TFIX or PFIX which will refer to the real address of the page and therefore need a page frame below 2 GB. For that reason page frames below 2 GB are assigned to pages whose page fault is raised internally.

The following shows the logic how page manager selects page frames above and below 2GB.

Type of page fault:

1. **External page fault (program exception x'11'):** select a 64-bit page frame
  - Search IPFQ64
    - Page frame found -> queue its PFTE at end of PSQ64 and use selected page frame.
  - IPFQ64 empty: search IPFQ
    - Page frame found -> queue its PFTE at end of PSQ and use selected page frame.
  - IPFQ empty: search PSQ64



- Scan PSQ64 for a certain number of elements.
  - R-bit on: requeue PFTE to end of PSQ64, reset R-bit.
  - C-bit on: enqueue PFTE for pre-page-out.
  - Use first page frame with R-bit off and C-bit off.
  - Page frame found -> queue its PFTE at end of PSQ64 and use selected page frame.
  - No free 64-bit page frame found: search PSQ
    - Scan PSQ for a certain number of elements.
    - R-bit on: requeue PFTE to end of PSQ, reset R-bit.
    - C-bit on: enqueue PFTE for pre-page-out.
    - Use first page frame with R-bit off and C-bit off.
    - Page frame found -> queue its PFTE at end of PSQ and reuse selected page frame.
  - None found: start unconditional page-out request.
  - NOPDS: system error if no page frame is available in IPFQ64 or IPFQ
- 2. Internally queued page fault (e.g. from TFIX): select a 31-bit or 24-bit page frame**
- Search IPFQ
    - Page frame found -> queue its PFTE at end of PSQ and use selected page frame.
  - IPFQ empty: search PSQ
    - Scan PSQ for a certain number of elements.
    - R-bit on: requeue PFTE to end of PSQ, reset R-bit.
    - C-bit on and IPFQ64 not empty: exchange page frame below 2GB with 64-bit page frame.
      - Chain PFTE of 64-bit page frame at bottom of PSQ64.
    - C-bit on and IPFQ64 empty: enqueue PFTE for pre-page-out.
    - Use first page frame with R-bit off and C-bit off.
    - Page frame found -> queue its PFTE at end of PSQ and reuse selected page frame.
  - None found: start unconditional page-out request.
  - NOPDS: system error if no page frame is available in IPFQ or in IPFQ64 for 'page-out'

---

## Page Handling Routines

The following conditions result in some form of page movement or reassignment of page frames and may require activity by the page manager (PMR) system task:

- Page Fault
- GETREAL request
- TFIX request
- PFI request
- PAGEIN request
- VIO POINT request
- LRA special operation exception

However, the PMR system task is not activated for the following requests:

- FREERREAL request
- TFREE request
- PFREE request
- RELPAG/FCEPGOUT request
- INVPAGE request

The requests that require the activity of the PMR system task are queued in the page-in queues or the page-out queue for the device on which the page to be handled resides.

For each device on which the page-data-set resides, control information is maintained in a Device Control block (DEVCB). The page-data-set devices are serviced in wrap around mode. The PMR system task tries to start an I/O-request on each device as long as requests are pending and not yet started.

One page-in queue exists for each partition/class and one for the system. In addition one page-out queue exists on each device. 'User-page-faults' (that means, page faults in the user area) are queued in the corresponding partition/class page fault queue; 'system-page-faults' (that means, all other page faults) are queued in the system page fault queue. Each queue consists of a forward chain of IOREs. For page-in requests the IOREs are the normal TIBs of the tasks waiting for completion of the page-fault handling and TIBSTATE (in TIB) contains the Extended Page Fault address of the page to be handled. For page-out requests pseudo TIBs are used which don't belong to any specific task and TIBAALU contains the address of the related allocation unit in the POSL. In case of unconditional page-out, TIBSTATE contains the address of the PFTE to be handled. Begin and end of chain are maintained per device in the DEVCB to allow for enqueue at the bottom and dequeue at the top of the queue. PFRQBEG indicates the header of the 1st page-in queue (PGQI), PORQBEG indicates the header of the page-out queue (PGQO).

The requests that require writing pages onto the page data set (it may be requested by GETREAL and for the handling of a page-fault) are queued in the page-out queue, and handled on a FIFO (first-in-first-out) basis.

There exist twenty-three pseudo TIBs (IOREs) for page-out (each 72 bytes long) which are allocated in the page management data area.

## Page Manager Processing

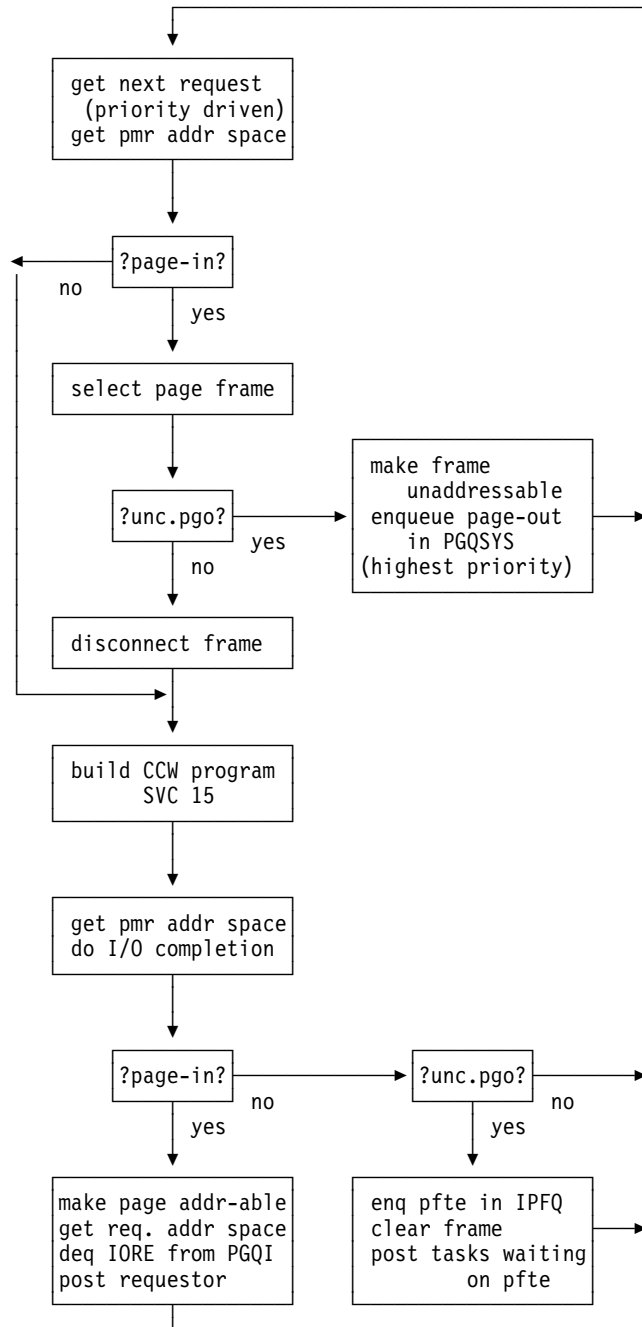


Figure 97. Page Manager Processing

As long as a PMR request is handled, the following fields in SGPDATA are set:

- PFDEVCB - address of actual device control block
- PFPGE - address of actual request element (IORE)
- PFSCB - address of SCB of area, the handled page(s) belong to
- PFVTABE - address of VTAB entry if page belongs to VPOOL, otherwise the field is negative

## Handling of a Page-In Request

A page-in request is enqueued to the proper page-in queue by the routine ENQUI. The PMR system task handles the page request queues in the priority order of the corresponding partitions. The system queue (PGQSYS) has the highest priority, the page-out queue (PGQO) has the lowest priority. Within each queue the entries are handled on a FIFO (first-in-first-out) basis.

The page manager (PMR) system task does the following steps when handling a page-in request.

- Select a page frame for the requested page (see Page Frame Selection and Pre-Page-Out) and remove it from the PSQ or the IPFQ.
- If the page frame selected is in use and its contents are not the same as that of the copy on the page data set (PDS) (that means, the change-bit is on) the page is set to the connected state and enqueued for page-out. If the selected page frame is in use and its contents are the same as that of the copy on the PDS, the page is disconnected.
- Read the requested page from the PDS, if a valid copy exists on the PDS (that means, PTEPDS bit is on in PTE) If not, the page is cleared to zero.
- Make the page addressable, that means: reset the reference and change bits, initialize the corresponding PFTE and enqueue it at the end of the PSQ.

## Handling of a Page-Out Request

Before a page-out request is actually enqueued for page frames below 2 GB the page manager checks the IPFQ64 for an available page frame. If there is an unused page frame the pages are swapped, and the page frame below 2 GB becomes available without page-out I/O.

A pre-page-out request is enqueued to the PGQO (page-out queue), by the routine ENQUOBLK, an unconditional page-out request on top of the PGQSYS (system page i/o queue) by the routines ENQUOUNC or ENQUOW.

The page manager (PMR) system task performs the following steps when handling an entry:

- Reset the change bit and set the PDS bit of the requested page.
- Indicate page-out as active in the PFTE and write the page onto the PDS.
- Reset the reference bit.
- If posting is required, post the tasks that are waiting for the page frame.
- Reset the PGQO indication in the PFTE.

Pre-page-out is done via blocked page I/O. That means the number of contiguous pages of the affected allocation unit is sufficient high. Whenever feasible unconditional page-out is combined with pre-page-out to do blocked page I/O whenever feasible.

**Note:** The handling of a pre-page-out request does not change the status of a page. After the completion of an unconditional page-out request the page or block is disconnected.

## Emergency Handling

Emergency handling is required if

IPFQ = <>  
& <pftte∈PSQ|pftteflg≠POABIT & R-bit(Page-frame(pfte))=OFF> = <>

that means, a page fault can't be handled immediately. This state is indicated by PMRFLAG.PMRIOWT and results in waiting on completion of some already started page-out requests. As long as PMRFLAG.PMRIOWT is ON, any page fault is queued into the related PGQI-queue but not yet processed (that means the page manager is not activated).

## LRA Exception Appendage

An LRA instruction will cause a special operation exception, when a page frame above 2 GB has been assigned to the referred page, and a 64-bit real address would have to be passed to the program which is in 31-bit addressing mode. When a special operation exception occurs for an LRA instruction, the program check handler calls the page manager appendage routine PMLRA64. The page manager then replaces the 64-bit page frame by a page frame below 2GB. Its real address is returned to the program instead of the 64-bit real address which caused the exception.

The routine PMLRA64

- searches for a page frame below 2 GB that is available for exchange,
- invalidates the page table entries of the virtual address (LRA operand) and target page frame,
- exchanges the data of the page frames,
- exchanges the storage keys including C-bit and R-bit of the page frames,
- restores the page table entries with the new page frame addresses.

The routine will always find a page frame below 2GB for exchange. If no available page frame is found below 2GB, the page I/O request queues are scanned for a request with a data area below 2GB which has not yet been started. That page frame is then exchanged with the 64-bit page frame that caused the exception. The number of unfixed pages will not be reduced by this action.

## Blocked Paging

### Blocked Paging Concepts

Blocked page I/O is triggered by the value of the actual allocation unit in the PISL and the POSL respectively. For page out the value is determined by the states of change bits of the addressable pages of the unit. For page in the value is determined by the states of pages of the unit to be read in. In any case, the value specifies the current blocking factor.

Usually pre-page-out will be started, if at least the minimal blocking factor is reached. This factor is dependent on the relation between the amounts of unconditional- and pre-page-outs. Both items are measured dynamically.

For any unconditional page-out request the related POSL state is inspected to check whether pre-page-out can be combined with the unconditional page-out request, even if the minimal blocking factor is not satisfied.

When servicing a PAGEIN SVC the page-in task supplies a PISL later be applied by the page I/O routines to do blocked page-in.

The POSL is managed by page selection and related routines.

If an I/O error occurs in a blocked page I/O request, the request is ignored or will be replaced by a sequence of unblocked I/O requests, which are handled as current.

Request Type	Actions
pre-page-out	increase error count undo (pte,change_bit,posl for failing pages) ignore error
blocked page-in	increase error count undo (pfte,pte) for failing pages reset IORE unblocked adjust pisl ignore error
blocked unconditl page-out	increase error count undo (pte,change_bit,posl) for all pages) reset IORE unblocked (incl PGQDELO-s) restart IORE

Figure 98. Handling of I/O Errors

PMR I/O errors related to data spaces result in canceling of the affected user task.

After establishing the related page manager address space, the requested processing routine - pginsio for page-in requests, pgoutsio for page-out requests respectively - gets control. Accordingly, at I/O completion time, the related page

manager address space is established and the processing routine - PGICIMPL for page-in, PGOCOMPL for page-out- gets control.

I/O error detection is done in the subroutine *PMRIOERR*.

### Blocked Page-Out

Pre-page-out activities are delayed until a complete block of pages (not a necessarily complete allocation unit) can be written onto the PDS.

The POSL is used to trigger pre-page-out of a block. If a page with change-bit equal to ON is detected at page frame selection time, the corresponding bit in the related POSL is set to ON. The POSL is addressed via the SCB, the bit in the POSL.PSLSTATE is identified by the EPA number of the affected page, the resulting byte describes the allocation unit. A POSL is provided for each private and the shared address areas and each data space. A PSL is not supported for the real address area (EXEC pgmname,REAL) and the private page manager address areas.

In the time delay between enqueueing blocked pre-page-out and starting the related CCW program one or more pages of this allocation unit might be invalidated or changed. Therefore, it is verified whether the blocking factor is still high enough to do the page-out. In case of page-out, the affected bit(s) in the POSL are reset before starting I/O otherwise the request is dequeued and the POSL remains unchanged.

**Note:** Whenever the change bit is reset the related POSL indication must be reset too.

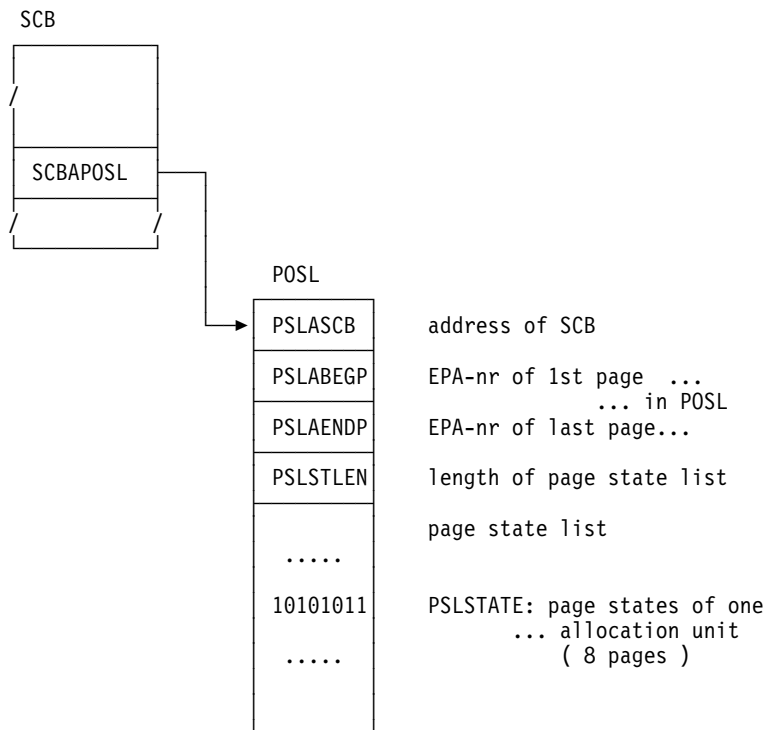


Figure 99. POSL

## Check and Control Blocked Page-Out

The value of POSL.PSLSTATE is applied as index in the table PREPGOTB to get the related PREPGENT entry. This entry provides all information necessary to check whether blocked page-out can be performed: to create the CCW-s at SVC-15 TIME. the number of contiguous pages in the current allocation unit is shown by PREPGENT.PREPGBLK and is compared with the actual minimal blocking value in SGPDATA.PPOBLCUR. If PREPGENT.PREPGBLK is equal to or greater than SGPDATA.PPOBLCUR, blocked pre-page-out is done and PREPGENT.PREPGVAL represents the "blocked" part.

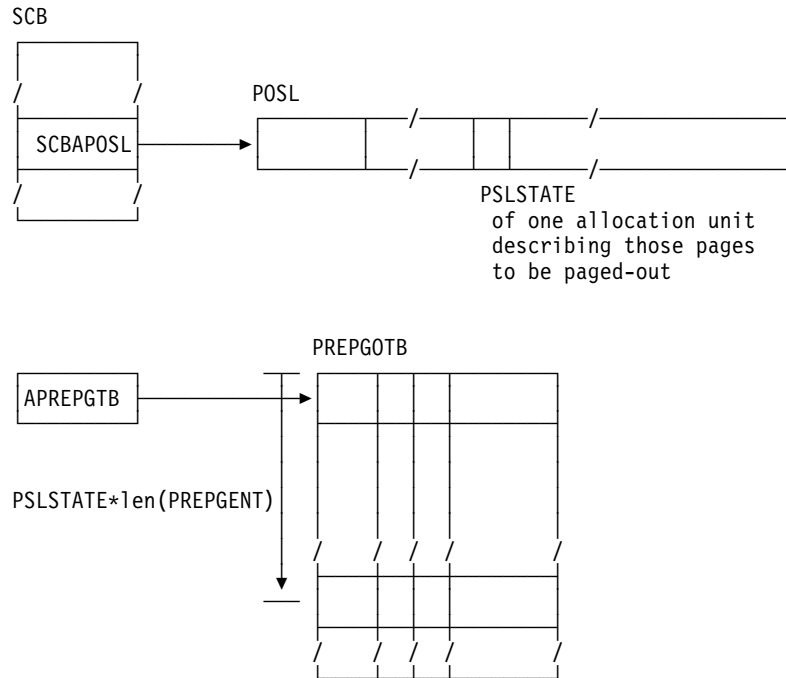


Figure 100. Relationship between POSL and PREPGOTB

The current value of the POSL.PSLSTATE is the index in PREPGOTB and determines the related PREPGENT entry:

## Page-Out Scenario

There are two different scenarios, one for blocked pre-page-out dependent on the minimal blocking factor and the other for unconditional page-out trying to do blocked page-out in any case.

## Enqueuing Pre-Page-Out Request

The blocking factor in the allocation unit is sufficient high and the allocation unit is not yet enqueued, an IORE is enqueued in PGQO. So far, no PFTE is affected.

DEVCB.DEVACT contains the address of the IORE with running I/O. If there is no active I/O at all for the DEVCB the field DEVCB.DEVACT is set to X'0'. The current value of POSL.PSLSTATE is saved into IORE.PGOEQPSL.



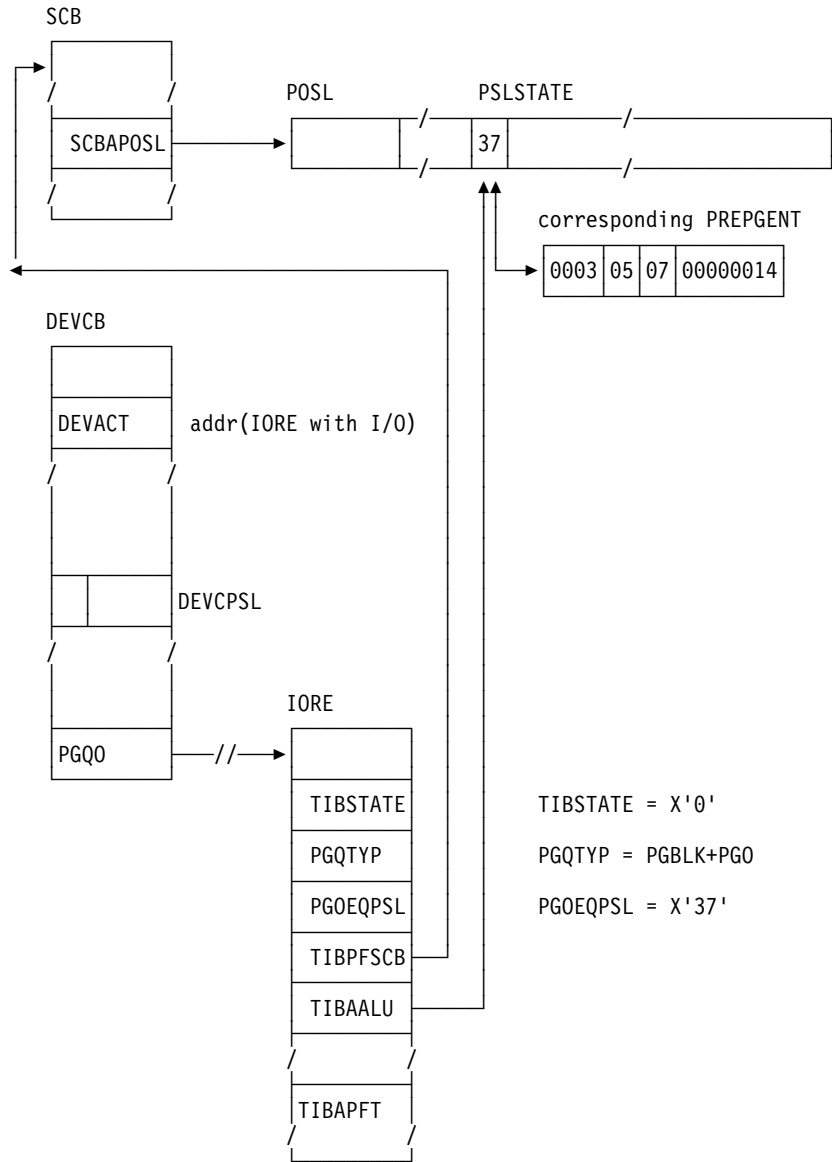


Figure 101. Enqueueing Pre-Page-Out Request

### Starting Pre-Page-Out Request

In the time interval between enqueueing the IORE and starting I/O some pages might be released. Accordingly, POSL.PSLSTATE of the allocation unit has been changed (for example, from X'37' to X'9D').

The value is used to get the corresponding PREPGENT entry, which contains the actual number of contiguous pages - equal to the actual blocking number - and the resulting blocking value applied to create the CCWs.

The affected ppte-s are connected to the SCB and to the IORE and are flagged in PFTEFLG with POEBIT=ON and POABIT=ON.

The resulting blocking value is saved in DEVCB.DEVCPSSL and IORE.PGOIOPSL.

POSL.PSLSTATE is updated accordingly (for example, the DEVCP SL value X'1C' applied on POSL.PSLSTATE X'9D' results in X'81').

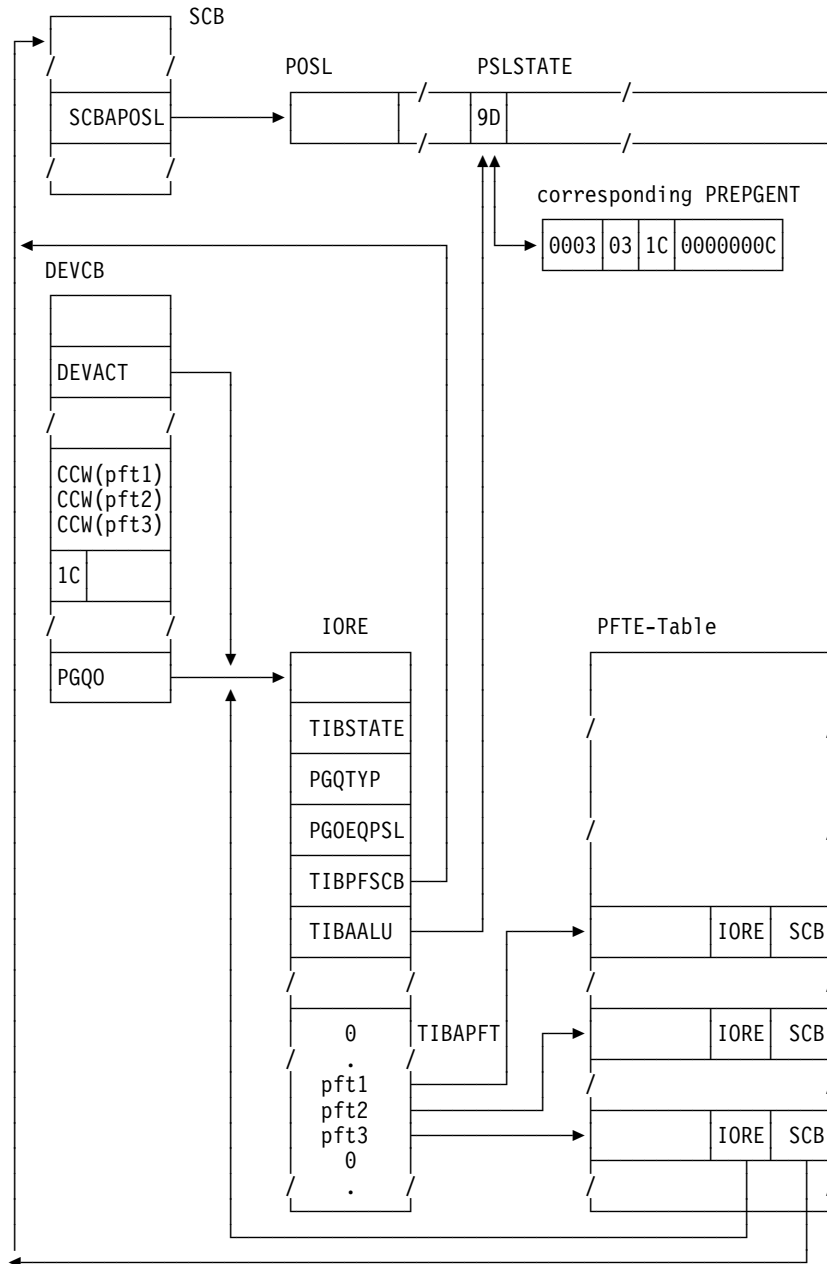


Figure 102. Starting Pre-Page-Out Request

The POSL.PSLSTATE is updated accordingly (for example, the DEVCPSL value X'1C' applied on the POSL.PSLSTATE X'9D' results in X'81').

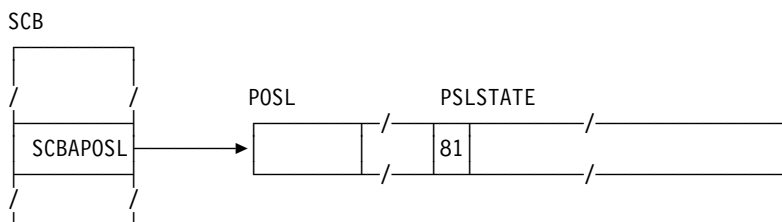


Figure 103. State of POSL after Starting the Request

### End of Cylinder Condition for CKD

If the CCW program would cross a cylinder boundary on a CKD device the CCW program will be cut on this cylinder boundary. The other pages will be handled by another I/O request at later time.

POSL.PSLSTATE and DEVCB.DEVCPSL must be adjusted accordingly (for example, EOC is detected for the third page with the POSL mask X'40', DEVCB.DEVCPSL is adjusted to X'18' and POSL.PSLSTATE to X'85'). Moreover, POEBIT and POABIT in PFTEFLG of not yet considered pfte-s must be reset.

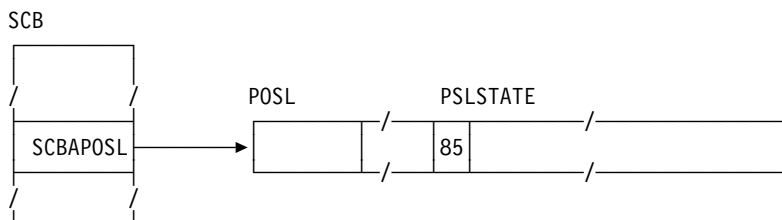


Figure 104. State of POSL after Adjusting for CKD

### Enqueuing Unconditional Page-Out Request

If the page selection algorithm has identified a frame containing page with change-bit=ON, this page must be written on the PDS before the frame can be used. POSL.PSLSTATE of the related allocation unit is set accordingly.

The request gets highest priority and the IORE is enqueued at top of PGQSYS of the corresponding DEVCB.

IORE.PGOEQPSL contains the related pslstate of the page to be page-out and POSL.PSLSTATE of the related allocation unit contains the current value including the poslmask of the unconditional page-out.

The selected PFTE is removed from the PSQ and PFTEFLG.POEBIT = ON.

The page to be page-out is set into disconnected state.

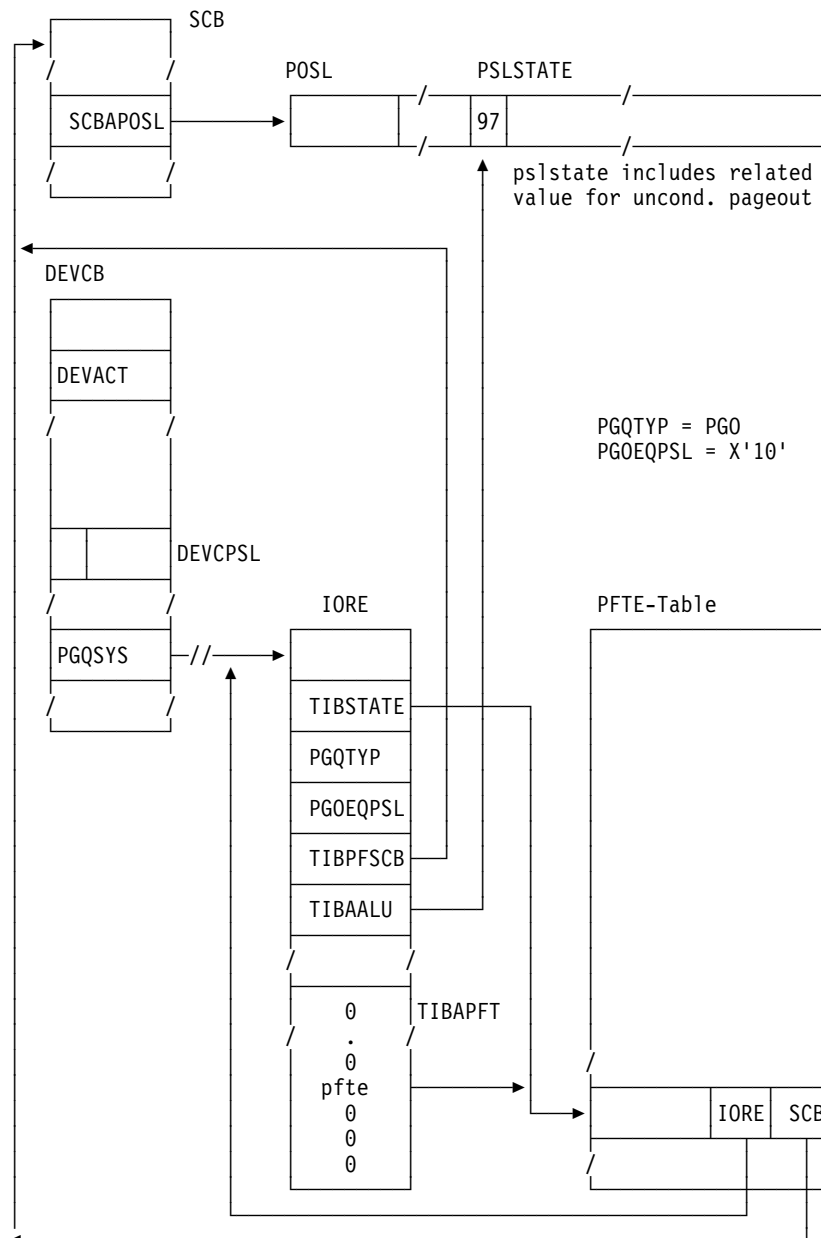


Figure 105. Enqueueing Unconditional Page-Out Request

### Special Consideration for PFTE with Active I/O (POABIT=ON)

Page-out runs concurrently to page frame replacement. As result a PFTE with active I/O might be selected. There are two possibilities:

- PFTE is part of a pre-page-out request  
as a consequence, the pre-page-out is modified to an unconditional page-out by updating IORE.TIBSTATE; the IORE remains in PGQO.
- PFTE is part of blocked unconditional page-out  
as a consequence, a new IORE is created with PGDELO indication and enqueued into the PGQSYS just behind the running page-out request.  
The IORE is needed in case of I/O error only.

### Determination of Blocking Value for Unconditional Page-Out

### **Special Consideration for PFTE with active I/O (POABIT=ON)**

Page-out runs concurrently to page frame replacement. As result a PFTE with active I/O might be selected. There are two possibilities:

- PFTE is part of a pre-page-out request  
as a consequence, the pre-page-out is modified to an unconditional page-out by updating IORE.TIBSTATE; the IORE remains in PGQO.
- PFTE is part of blocked unconditional page-out  
as a consequence, a new IORE is created with PGDELO indication and enqueued into the PGQSYS just behind the running page-out request.  
The IORE is needed only if an I/O error occurs.

### **Determination of Blocking Value for Unconditional Page-Out**

It is most effective to combine the unconditional page-out request with other page-out requests for the same allocation unit. Analogously to pre-page-out, the POSL of the related allocation unit might be changed between enqueueing and starting the I/O request. Therefore, the determination of the blocking factor and the creation of the CCW-program is done at SVC-15 time. The new POSL.pslstate value of the allocation unit is inspected by means of the POSL mask of the unconditional page-out. The resulting blocking value is determined as follows: every POSL mask of an allocation unit is related to a special control table (UNCPTBxx). Accordingly to the POSL mask of the unconditional page-out the corresponding control table is selected. By the way, there are eighth special control tables. The current POSL value is applied as index in this control table UNCPHOxx to get the effective POSL value which is used as index in the page-out control table (PREPGOTB) to get the actual PREPGENT entry.

Now all information is available to create the CCW program.

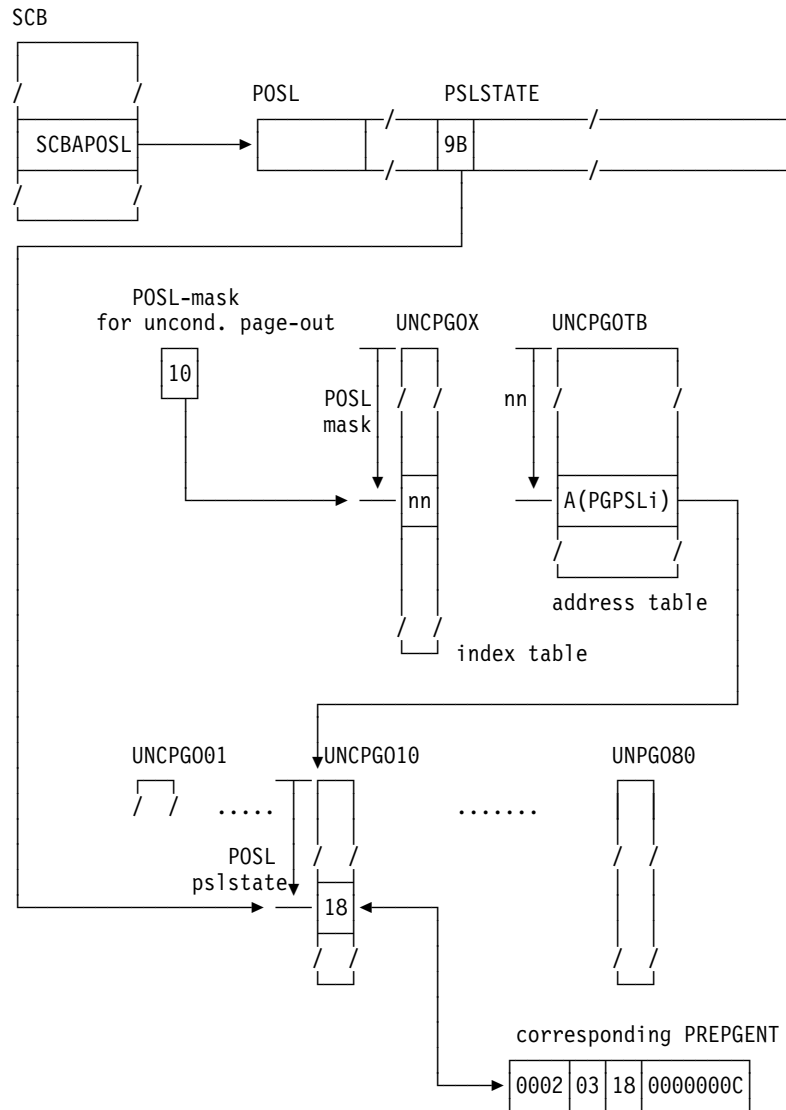


Figure 106. Determination of Blocking Value for Unconditional Page-Out

### Starting Unconditional Page-Out Request

In the time interval between enqueueing the IORE and starting I/O some pages might be changed. Accordingly, the state of the allocation unit has been changed (for example, from X'97' to X'9B'). Whenever possible, blocked page-out will be done, even if the current blocking value does not satisfy the value in SGPDATA.PPOBLCUR.

### Special Considerations for Combined Unconditional Page-Out

A blocked unconditional page-out might contain the PFTE-s of other unconditional page-outs also enqueueing in the PGQSYS of the same DEVCB. Because every unconditional page-out is represented by its IORE, the not yet started IORE(-s) are indicated by PGQTYP.PGDELO. (The IORE is needed in case of I/O error incl. end-of-cylinder).

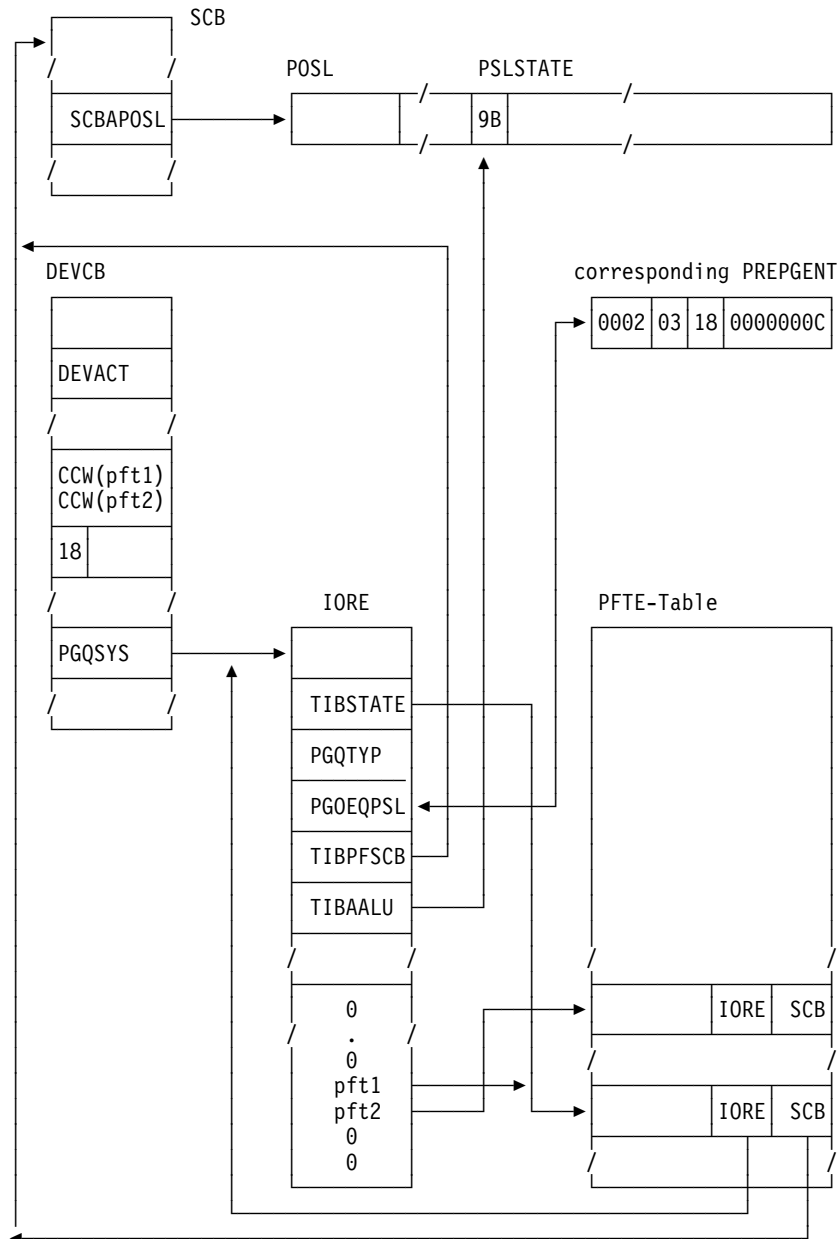


Figure 107. Starting Unconditional Page-Out Request

## Tuning of Pre-Page-Out

The objectives are: minimize the number total page-outs and minimize the number of unconditional page-outs. There are two threshold values:

1. No pre-page-out at all, that means only unconditional page-out
2. No unconditional-page-out at all, that means only pre-page-out

The number of pre-page-out is a function of both the blocking factor  $b$  and the searching depth  $k$  in the PSQ (performed by page selection). An increasing value of  $k$  or a decreasing value of  $b$  does result in an increasing number of pre-page-outs. A high rate of pre-page-outs results in a low rate of unconditional page-outs; but it might be a high rate of useless pre-page-outs.

The tuning algorithm applies these dependencies: pre-page-out and unconditional page-out rates must be balanced; additional parameter is the number of PDS devices.

```

procedure pgotunig(k : inout  $\mathbb{N}$ , b : inout  $\mathbb{N}$ ) :=
    /* k : cur searching depth */
    /* b : cur blocking factor */

    dcl
        d  $\in$   $\mathbb{N}$ ,          /* no of PDS devices      */
        m  $\in$   $\mathbb{N}$ ,          /* maximal blocking factor */
        n  $\in$   $\mathbb{N}$ ,          /* minimal blocking factor */
        u  $\in$   $\mathbb{N}$ ,          /* exp.average of unc-pgout*/
        o  $\in$   $\mathbb{N}$ ,          /* exp.average of pre-pgout*/
        f  $\in$   $\mathbb{N}$ ,          /* constant                */
        h  $\in$   $\mathbb{N}$ ,          /* maximal value for search*/
        l  $\in$   $\mathbb{N}$ ,          /* minimal value for search*/
        /* ... ing depth      */
    dclend;

    if (u = 0) & (b < m)
        then b = b + 1;          /* decrease pre-page-out */
        else do;
            if (2*u > d)
                then do;          /* increase pre-page-out */
                    if k < h then h = k + 1;
                    if b > n then b = b - 1;
                end;
            else do;
                if (o < u) & (b > n)
                    then b = b - 1;          /* increase pre-page-out */
                    else do;
                        if (o > d)
                            then do;          /* decrease pre-page-out */
                                if k > l then k = k - 1;
                                if b < m then b = b + 1;
                            end;
                        end
                    end
            end
        end;
    end pgotunig;

```



## Blocked Page-In

Applicable for the supervisor services (SVC X'57' and SVC X'121').

During the first scan of the parameter list in procedure *PAGEIN* (supervisor generation macro *SGPOPT*) all affected pages in the PSQ are removed to bottom of the PSQ. A further scan creates a PISL for those pages not currently addressable and with a valid copy on PDS. For each allocation unit, blocked page-in will be tried. A third scan is necessary if blocked page-in was not successful and page-in is provided via normal page fault processing.

Blocked page-in is done per allocation unit if:

- there are contiguous pages in the allocation unit with *ptepds=ON* and
- $\text{len(IPFQ)} \geq (\text{number of pages with PDSBIT=ON})$ .

In more detail this is:

1. The PISL is initialized for each allocation unit.
2. If the page is not addressable and there is a valid copy on PDS the related *PISL.pslstate* is set to ON.
3. The procedure *PGINENQI* checks whether blocked page-in can be performed and if so, enqueues the IORE in the *PGQSYS* of the related *DEVCB* and sets the Page-in system task into wait (*PMRBND*).
4. At *SVC-15* time the *CCW* program for blocked page-in will be generated by the procedure *PGINBLK*.
5. I/O completion is done in the procedure *PGICOMPL* and the waiting Page-in system task is posted.

## Page In State List (PISL)

The page-in system task owns the PISL located in *SGPDATA* and describing the page states of one allocation unit. Each bit in *PISL.PSLSTATE* identifies one page. The bit is set to ON if the related page must be read in.

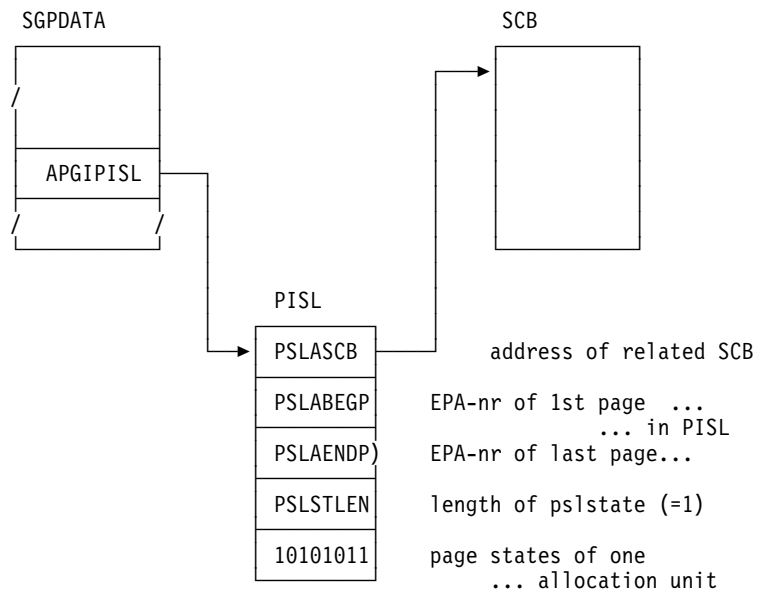


Figure 108. PISL

## Check and Control Blocked Page-In

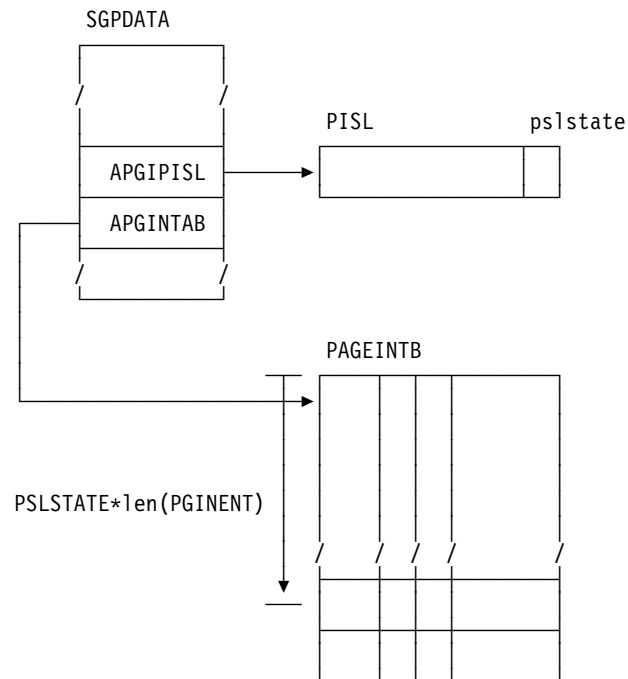


Figure 109. Relationship between PISL and PAGEINTB

The value of PISL.PSLSTATE is applied as index in the table PAGEINTB to get the corresponding PGINENT entry.

### Page-In Scenario Enqueuing Page-In Request

Blocked page-in is done for the allocation units one after the other. For each allocation unit the prerequisites are:

- there are valid pages on the PDS and
- there are sufficient free frames to satisfy the page-in request.

During the page-in service the PISL is built and initialized for each allocation unit. Every page of the current allocation unit is inspected to decide whether a page-in must be done. If so, the related PISL mask is activated in the PISL and the page is connected to a free frame. After inspection of the allocation unit, the page-in request is initiated by enqueueing the IORE in the PGQSYS of the corresponding DEVCB.

The field DEVCB.DEVACT contains the address of the IORE with not yet completed I/O. If there is no I/O running for this DEVCB, DEVCB.DEVACT contains X'0'.

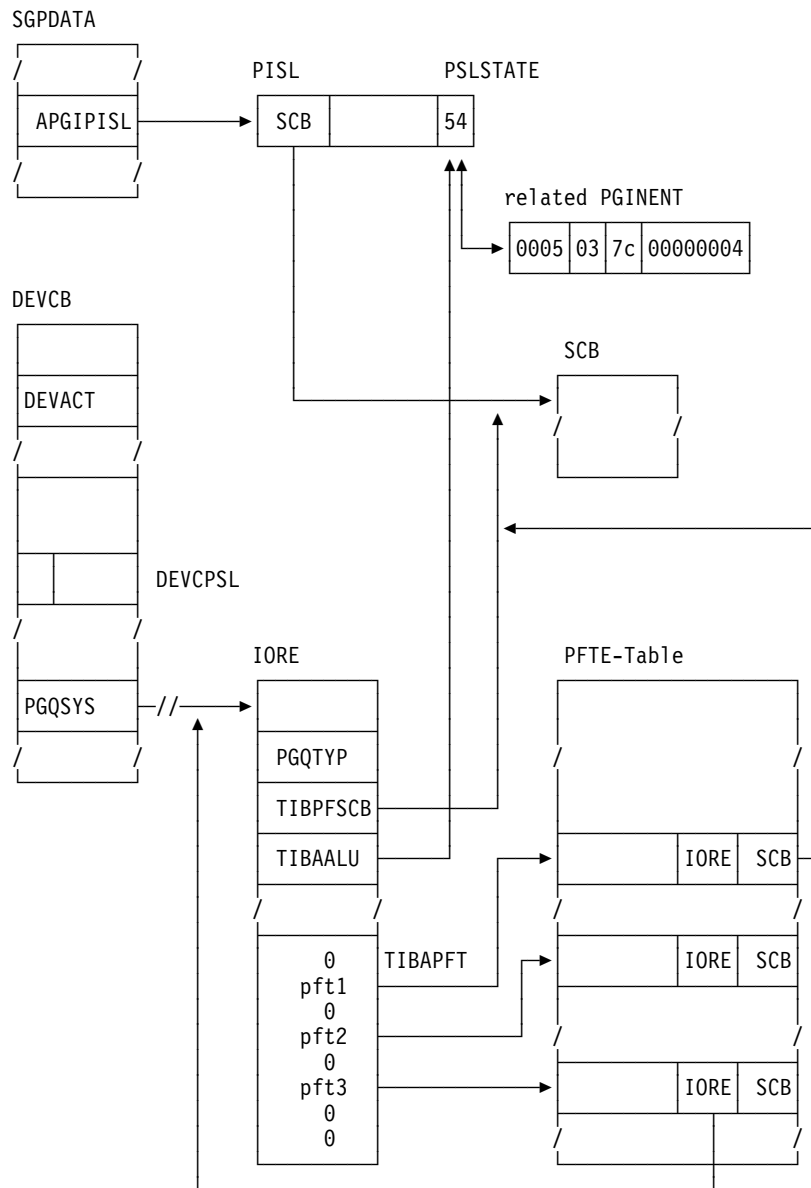


Figure 110. Enqueueing Page-In Request

### Starting Page-In Request

Between enqueue and SVC-15 time no change of the PISL value is possible. The pages are connected and the frames are reserved.

At SVC 15 time the request is set up accordingly to the PISL value. Analogously to blocked page-out the control table PAGEINTC provides information about the blocking factor and the effective blocking etc which is used to create the CCW program. All affected pages of an allocation unit are paged-in via one CCW-program, any gap between affected pages are handled by one or more Read CCW(-s) with data transfer suppression.

The field PISL.PSLSTATE is reset to zero.

The corresponding pte-s are still in connected state.

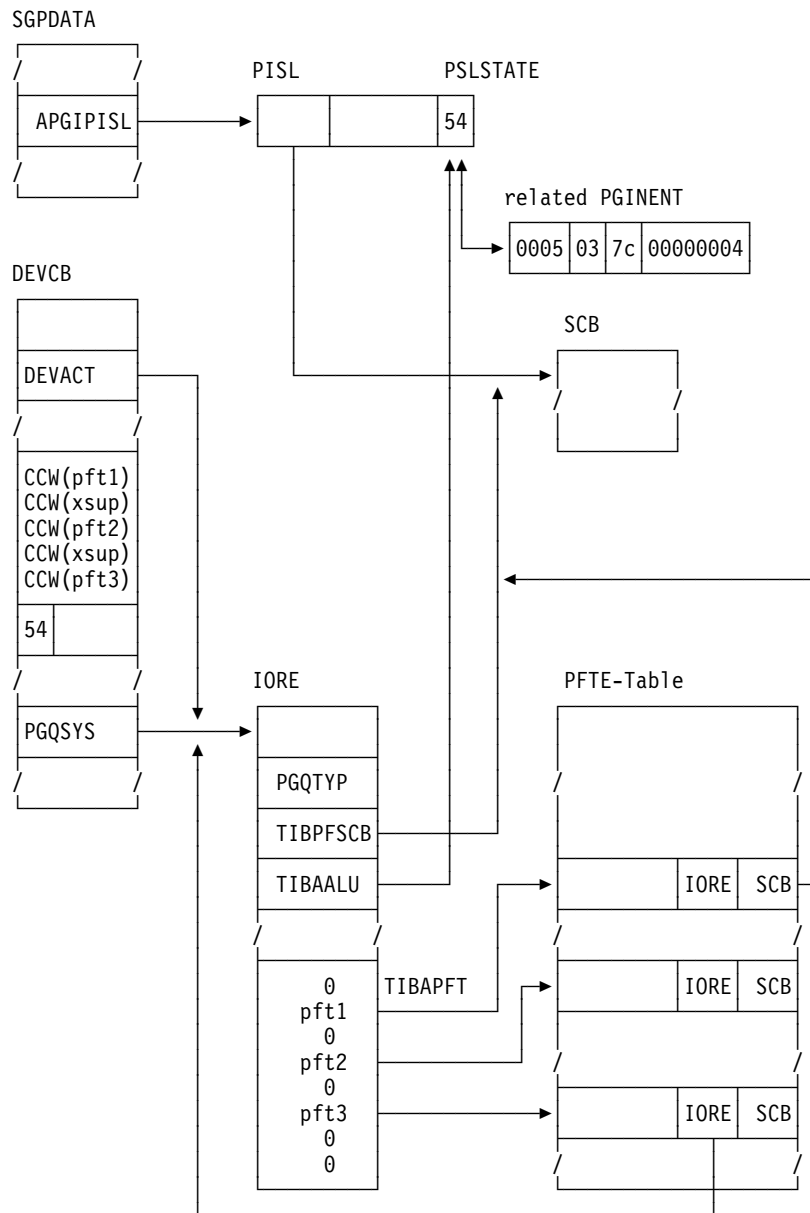


Figure 111. Starting Page-In Request

## Page Fault Handling Overlap

Programs that execute in virtual mode and do their own multi-tasking can use the page fault handling overlap facility. This gives the user the opportunity to control the page-in queue entry for the page fault caused by its own task. This is done by a user-written page fault appendage routine.

Whenever a page-fault occurs, page management first checks if a page fault appendage has been initiated for the task and the type of the page fault (the user can specify, that he doesn't want to overlap page-faults due to access to a data-space).

If the task has an appendage for the actual page-fault, control is first passed to that appendage, unless the task is using a supervisor service, the LTA, or an ACF/VTAM function. The request is then enqueued in the page fault queue using a

special TIB (PHOTIB) located in the PCB, or it gives an indication that a page fault is already pending for that task. The task causing the page fault is not set into the wait state.

When pseudo-page faults occur (running under VM and 'SET PAGEX ON' issued), the page fault overlap handling appendages are not entered.

If the page fault was caused by a supervisor service or logical transient, or if an ACF/VTAM function is outstanding, or if the user doesn't want to overlap page-faults in data-spaces, no overlap is performed. The page fault is handled like any normal page fault condition and the task is set into the wait state.

When an asynchronous page fault has been handled, the appendage is entered again to see if there are any more page faults to be processed. If so, the page-in request returned from the appendage is enqueued in the correct device queue.

## **Pseudo-Page Fault**

Pseudo page faults are a special type of program check used when running under VM. There are two different types of pseudo page faults:

- Pseudo page fault exception (whenever VM gets a page fault and must do I/O operations)
- Pseudo page fault completion (whenever the I/O operation of VM is completed)

For both exceptions VM passes control to VSE by means of program check interruption.

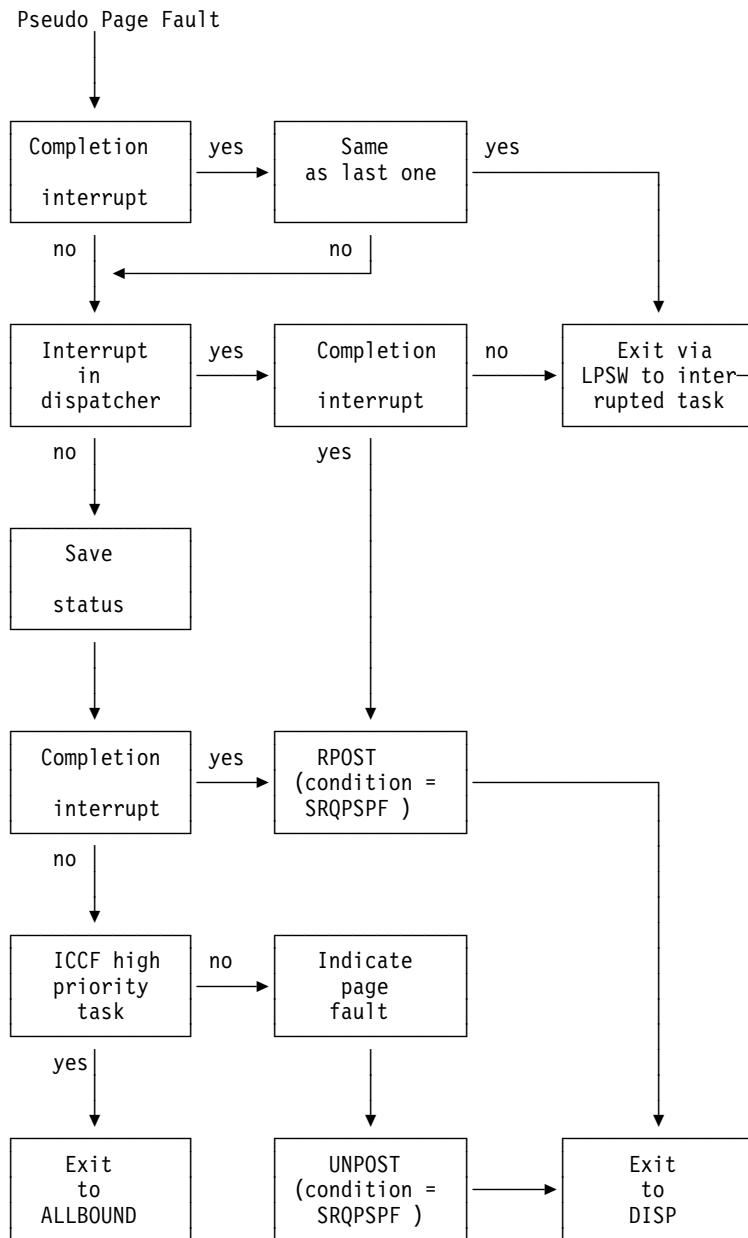


Figure 112. Pseudo Page Fault Handling

## GETREAL Request

A GETREAL request is issued by SVC X'37' (request for SDAID area), SVC X'3A' (if initialization of a real partition is requested) and GETPT service, to reserve an area of real storage.

On entry, register 2 contains the beginning, and register 3 the end address of the area requested. All PFT entries of the page frames in this area are posted as not fixable (NFRP-bit is set on), and the TFIX counter of each entry is checked for zero (page is not TFIXed). If a page frame is found to be TFIXed, the requesting task is set to PGFX bound. If the area requested is free of TFIXed pages, the following steps are executed:

1. If the page frame is unused:

Remove the PFTE from IPFQ. Increase PFIx counter in PFTE by 1. If it is not a special GETREAL request from GETPT (GTRSBIT not on in PCB.FIXTYPE) the following is done in addition:

- a. Make page with same address as page frame addressable.
  - b. Reset the NFRP bit, and increase the partition PFIx counter by 1.
2. If the page is connected to a page frame:  
Wait for end of page connected state and take actions depending on the new state of the page frame.
  3. If the page frame contains a valid page that is requested by PFIx:  
Get an unfixed page frame and exchange the contents of the two page frames. Take actions according to the new state of the page frame.
  4. If the page frame contains a valid page which is not in connected state and which is not requested by PFIx:  
Remove the PFTE from the PSQ. If the change bit for the page frame is set, call ENQUOW to write the page onto the PDS. On return, take actions depending on the new state of the page frame.  
If the change bit for the page frame is not set, the page is disconnected using routine PAGDISCA and the actions described in 1. are taken (except for the removing of the PFTE from its queue).
  5. If a page frame is found to be unusable because of a hardware error (DRAP bit in PFT entry on):  
No area is allocated when this condition is detected in the first page frame. If the page frame in error is not the first one, the allocated area ends at the start address of the failing page frame, if it is a SVC X'37' request, otherwise no area is allocated.

The following return codes are passed by GETREAL:

- 0 = The requested area is reserved (PFIxed).
- 4 = The page frame belongs to failing storage and is not the first page of the real partition.
- 8 = The page frame belongs to failing storage and is the first page of the real partition.

## TFIX Request

The TFIX routine fixes pages temporarily, that is, a page is fixed in a page frame for the duration of an I/O operation. This routine is called by the CCW translation routines, the Fetch routine, the SVC X'2C' routine, and others.

The caller provides in register 1 an address that points to a parameter list of the following format:

Parmlist := list(Parm)

Parm := (addr. in first page to be handled BIN FIXED(31), addr. in last page to be handled BIN FIXED(31))

Parm := ( A(0), NIL) indicates end of list.

A TFIX request for p pages which are not already TFIxed or PFIxed can be satisfied as long as the condition

$$p \leq \text{NPSQE} - \text{MINPSQE}$$

is satisfied. Otherwise the requesting task is set into wait. NPSQE is reduced by p pages:

$$\text{NPSQE}' = \text{NPSQE} - p$$

**Note:** Here and in the following formulas the new value of the variable xxxx is noted as xxxx', the original value is noted as xxxx.

The length of PSQ is reduced by q page frames:

$$\text{len}(\text{PSQ})' = \text{len}(\text{PSQ}) - q$$

where  $q = p - r$  with  $r = \min(\text{len}(\text{IPFQ}), p)$ .

Analogously yields:

$$\text{len}(\text{IPFQ})' = \text{len}(\text{IPFQ}) - r$$

The TFIX counter of all PFTEs is increased in any case:

$$\text{PFTE.TFIXC}' = \text{PFTE.TFIXC} + 1$$

The following return codes are passed by the TFIX routine:

- 0 = If the request is issued by the fetch routine and the number of available page frames in the PSQ reaches a minimum, and no page can be TFIXed; or  
if the request is not from the Fetch routine and the number of available page frames in the PSQ reaches a minimum, and no page can be TFIXed.
- 4 = The TFIX counter has reached the maximum value for a page and the page cannot be TFIXed.
- 8 = All requested pages are TFIXed.

A special interface is established for a TFIX request from the Fetch routine. It has to be ensured that at least 2 pages can be fixed. As long as  $\text{NPSQE} > \text{MINPSQE} - 2$ , the Fetch request is satisfied. If not all requested pages can be fixed, control is given back to the Fetch routine without freeing the pages already fixed for this request.

## PFIX Request

A PFIX request may be issued by a user task or by the restart (RSTRT) statement processor (Job Control). Actually it is called by SVC X'43', SVC X'4A', SVC X'79', GETPT service, XPCC service, FETCH and storage management.

Register 1 points to a parameter list that defines the pages to be PFIXed. If the page is not in storage the request is enqueued to the page queue and the PMR system task is activated.

A PFIX request for p pages can be performed immediately, as long as the conditions

- $p + \text{SMPFIX} < \text{SMAXPFIX} + 1$
- $p < \text{NPSQE} - \text{MINPSQE} + 1$
- $\text{PFTE.PFIXC} < \text{MAXPFIX}$  for all PFTEs associated to the PFIXed pages
- the pages occupy frames belonging to the correct area (1, 2 or 3).

are satisfied. Otherwise the requesting task is set into wait or is posted with a return code indicating no PFIX possible.



Before a page can be fixed it must be determined whether this can be done immediately or not. If the page occupies a page frame in the PFIX area\_1 or PFIX area\_2 respectively, the page can be fixed at once. If the page occupies a page frame in PFIX area\_3 and the PFIX macro is specified with RLOC=ANY, the page can be fixed at once. If the page occupies a page frame outside the areas or if the page is not yet addressable, a page frame out of the related PFIX area must be selected.

The page frame table entry address of this reserved page frame is stored in the partition control block (label PFTERSVD) by the PFIX routine, or by the PFREE routine or by the TFREE routine, if a page has to be freed before the page frame can be reserved.

If there is no page frame for a PFIX request in PFIX area\_2 but there are pages only TFIXed, all page frames in the related PFIX area are set to 'not temporarily fixable' and the task is put into the wait state; processing of the request continues as soon as a page has been freed by either a TFREE or by a PFREE request.

If there are no page frames for a PFIX requests in PFIX area\_1 or area\_3 the requesting tasks are put into wait state of the related area. There are frames occupied by TFIXed pages and the processing is resumed as soon as a page has been freed by either a TFREE or by a PFREE request.

If the page was neither PFIXed nor TFIXed, the corresponding page frame table entry is removed from the page selection queue, NPSQE is decreased by 1, and the partition PFIX counter is increased by 1.

The state changes for NPSQE etc. are the same as for TFIX. The PFIX counter of all PFTEs is increased:

$$\text{PFTE.PFIXC}' = \text{PFTE.PFIXC} + 1$$

The pages are PFIXed one after each other and if during this process the free frames are exhausted, all pages which have just been PFIXed are freed again. A special return code, is passed to the requesting task, indicating that the PFIX request cannot be performed under the actual system conditions.

The page will be fixed immediately, if the page is in real storage and if the following conditions are true:

- The page frame is in the correct real partition. In that case, it is only necessary to increase the PFIX counter by 1 and to remove the page frame from the selection pool if it has not already been removed.
- The page is not TFIXed and the page frame is not in the correct real partition, but a page frame in the real partition is available for PFIXing. The two pages are then exchanged and the page is PFIXed.

The following return codes are passed by the PFIX routine:

- 0 = Function successfully completed.
- 4 = Maximum number of allowed PFIXed pages for the partition is exceeded by this request only.
- 8 = Maximum number of allowed PFIXed pages for the partition is exceeded because of previous PFIX requests.
- 12 = Negative length of area or invalid address.

- 16 = PFIxed page above 16MB found, but current request with RLOC=BELOW.
- 20 = Invalid function code or option (not possible if macro interface is used).
- 24 = Wait for TFREE required, but caller requested return instead of WAIT. Only possible on subroutine interface.
- 28 = PFIx counter overflow. Only possible on subroutine interface.

### **PFIx Requests for RSTRT**

Handling PFIx requests for the RSTRT routine (Job Control) requires special action because each PFIxed page must be returned to the page frame in which it was located at the time the program was checkpointed. When a page is PFIxed by the RSTRT processor, not only the page address but also the page frame address and the value of the PFIx counter are passed. The address of the reserved page frame is placed in the field PCB.PFTERSVD for the task; the page is PFIxed in the reserved page frame and thereafter the PFIx counter of the page is set to its value at checkpoint time.

## **PAGE-IN Request**

A valid page-in request is handled by the PGIN system task, which is activated when the SVC X'57' routine has received such a request. The task's dispatching priority is higher than that of the Fetch (SUPVR) task, but lower than that of the page manager (PMR) system task.

The PGIN task runs asynchronously with the requesting user task.

For a page in real storage, the task determines (by looking at the corresponding PFT entry) whether this page is fixed.

- If the page is fixed, the request for the page is ignored.
- If the page is not fixed, its reference bit is set and the associated page frame is enqueued at the end of the Page Selection Queue (PSQ).

For a page not in real storage, the PGIN system task uses the ENQUI routine to have this page enqueued to the page-in queue. The request is then handled like a page-in request that resulted from a normal page fault; however, no exit is taken to a private routine that may be specified in a SETPFA macro in the program which issued the page-in request.

The PGIN system task detects the following error conditions and takes the actions indicated:

- If a page is outside the partition in which the requesting program is executing, the request for that page is ignored.
- If an area specification contains a negative length, the request for that area is ignored.

The task posts an ECB (if one is specified) as shown for SVC X'57' in "Supervisor Call Interrupt (SVC)" on page 27. The ECB's address is obtained from the currently processed PAGETAB entry.

Whenever a task is terminated, the scan routine SCANPGT scans table PAGETAB and deletes all entries that carry the task's TID. If the PGIN system task is processing a page-in request of a task which is being terminated, the PGIN system task stops processing of that page-in request.

## TFREE Request

A TFREE request is issued by routines such as CCW translation, SVC X'2C' or Fetch, to release TFIxed pages.

Register 1 points to a parameter list that defines the pages to be freed (see description of TFIx).

The TFREE request frees  $p$  page frames and the TFIx counter of all affected PFTEs is decreased:

$$\text{PFTE.TFIxC}' = \text{PFTE.TFIxC} - 1$$

Only if the conditions

$$\begin{aligned}\text{PFTE.TFIxC}' &= 0 \\ \text{PFTE.PFIxC} &= 0\end{aligned}$$

are satisfied for  $q < p + 1$  PFTEs, the  $q$  related page frames can be used by the page replacement algorithm or for PFIx / GETREAL requests. The page frames are inserted in the PSQ; that means:

$$\begin{aligned}\text{NPSQE}' &= \text{NPSQE} + q \\ \text{len(PSQ)'} &= \text{len(PSQ)} + q\end{aligned}$$

Additionally, the tasks waiting for free page frames must be posted if  $\text{NPSQE}' > \text{MINPSQE}$ .

Depending on the setting of bits NFRP and NFVP in the PFTE, additional actions may be taken when returning the PFTE to the PSQ:

NFVP=ON:

The freed page is requested by PFIx but the page frame does not belong to the real partition. The task registered in the PCB (FIxTIB) of the partition that is identified by the PFTEPIK field in the PFTE is posted ready to run.

NFRP=ON:

The freed page frame is requested by PFIx. The address of the PFTE of the freed page frame is inserted in PFTERSVD of PCB (see SVC X'43') and thus reserved for the PFIx request. The task issuing the PFIx request is posted ready to run. All page frames in the partition, except the reserved one, are set to temporarily fixable (NFRP=OFF) before the next request is processed.

## FREEReAL and PFRReE Requests

The PFRReE request frees  $p$  page frames and the PFIx counter of all affected PFTEs is decreased:

$$\text{PFTE.PFIxC}' = \text{PFTE.PFIxC} - 1$$

The conditions for further processing and the processing itself is analogous to that one of TFReE.

For handling of FREEReAL and PFRReE requests see SVC X'36' (FREEReAL) and SVC X'44' (PFRReE) in "Supervisor Call Interrupt (SVC)" on page 27.

## RELPAAG and FCEPGOUT Requests

For the handling of RELPAAG and FCEPGOUT requests see SVC X'55' (RELPAAG) and SVC X'56' (FCEPGOUT), in "Supervisor Call Interrupt (SVC)" on page 27.

## INVPAGE Requests

The INVPAGE service is used to set a number of virtual pages to disconnected with no copy on page-data set (for  $R5 \geq 0$ ) or to invalid (for  $R5 < 0$ ). In addition for  $R5 \geq 0$ , the storage key provided in R5 is set for the area. The page table and page frame table entries belonging to specific pages are initialized and the allocation and deallocation of blocks on page-data set is done (using PTASG and PTUSG routines).

The following parameters are passed to this routine:

RD: Address located in the first page of the area to be invalidated.

R1: Address located in the last page of the area to be invalidated.

R5:

positive or zero: storage key for disconnected pages.

negative: area to be deactivated.

RE: Return address

RB: Address of data space SCB (only if entered for data spaces)

If the area to be invalidated belongs to an active virtual partition or data space (i.e.  $R5 \geq 0$ ) the corresponding page table entries (PTE) are set to X'00KP0400' where K corresponds to storage key and where P indicates whether the page is fetch-protected ( $P=8$ ) or not ( $P=0$ ). If the area to be invalidated belongs to an inactive part of a virtual partition or data space (i.e.  $R5 < 0$ ) the corresponding PTE's are set to X'00002400'. Each PTE within the area defined by RD and R1 is initialized in that way. If the page referred to by an entry is in processor storage, the page frame table entry of the corresponding page frame is initialized as follows:

- The page frame is marked as unused (the PNRINV bit in S370FLG is set), and the PFI counter is set to zero.
- The page frame is removed from the page selection queue and enqueued to the top of the invalid page frame queue.
- The page frame is cleared.

## VIO POINT Request

The VIO storage is considered as an extension of the page data set. The size of a VIO storage block is equal to the size of a page. To control the VIO storage a number of pages in the address space is reserved for system usage. This area is named V-POOL and is located at the end of the SVA (24-bit) area.

As a result of a VIO point request, the user gets access to a page out of V-POOL, which contains the requested block of his VIO-file. The next VIO POINT request frees implicitly the block obtained by the previous request (that means, the user is no more allowed to access it directly).

The system tries to keep as much VIO-blocks as possible in real storage. Therefore, if a block is freed it is not immediately written to page data set but the page representing the block is set in connected state instead. If a page is requested by a VIO POINT request and no free page exists in V-POOL, an available V-POOL page is freed by disconnecting the page and setting the frame in 'block-connected' state (i.e. the PFTEBLK bit is set on in the corresponding PFTE).

The page frames occupied by VIO storage blocks are written on the PDS due to paging.

VIO storage is allocated in units of 64K. The total VIO storage is represented by an allocation string pointed to by VIOCM.VIOSPBEG. One byte in this string represents 64K of VIO storage (X'FF' indicates an occupied segment of 64K and X'00' indicates a free segment). VIOCM.VIOSPEND points to the last byte of the allocation string.

The VIO storage is managed using the following tables:

- VTAB (V-POOL table) which contains one entry per page in V-POOL, see Figure 198 on page 458.
- BLKTAB (block table) which contains one entry per block of VIO storage, see Figure 200 on page 460.
- VIOTAB (vio identification block) one VIOTAB entry exists per open VIO-file, see Figure 199 on page 459.
- FLSEGTBE (file segment-table) contains up to 8 segment table entries for a VIO file.

One segment table entry is 2 bytes long and contains the total blocknumber of the first block belonging to this segment. If more than 8 segments are required, a new FLSEGTBE is queued to the existing one.

Handling of VTAB-entries (VTABEs):

Two queues are maintained to handle the VTAB-entries. One, the free queue, contains all VTABEs which are not connected to a VIO storage block (VTUSCNT < 0). The other, the available queue, contains all VTABEs which are connected to a VIO storage block, but the user is not allowed to access it directly (VTUSCNT=0). VTABEs which are active that means, the user is allowed to access the page represented by the entry (VTUSCNT > 0) are not queued.

To allow enqueue at the bottom and dequeue at the top of the available queue, begin and end of this queue is maintained. For the free queue only begin of queue is maintained.

If due to a free the VTUSCNT reaches zero, the VTABE is enqueued on the bottom of the available queue. If a free VTABE is requested and the free queue is empty the first entry in the available queue is freed.

**Note:** As long as VTUSCNT ≥ 0, the PTE belonging to the VPOOL-page of the VTAB entry represents the status of the block (i.e. BLKSTAT=PGCON); or as long as BLKSTAT=PGCON, the PTE represents the status of the block.

**Valid states in VIO:**

```

IF BLKSTAT=PGCON /* page connected to block */
  THEN VTBLKN(BLKPAGE)=BLKN
      VTUSCNT>=0
      BLKPAG e1. VPOOL
      IF VTUSCNT=0
        THEN PAGSTAT=CON|DISC
          IF PAGSTAT=CON THEN PFSTAT=(ADDR or CON)
      IF VTUSCNT>0
        THEN PAGSTAT e1. (ADDR,CON,DISC)
          IF PAGSTAT=ADDR
            THEN PFSTAT=ADDR
          IF PAGSTAT=CON
            THEN PFSTAT=CON

IF BLKSTAT=FRCON /* only frame connected to block */
  THEN BLKPAG = addr of frame connected to block
      PFTEEPA#(BLKPAG) = total block number of connected block
      PFSTAT(BLKPAGE) = (ADDR and PFTEBLK) or (CON and PFTEBLK)

IF VTUSCNT>0 /* VPOOL page in use */
  THEN BLKSTAT(VTBLKN)=PGCON
      BLKPAG(VTBLKN)=VPAG
      IF PAGSTAT=ADDR
        THEN PFTEEPA#(PTEFRA)=VPAG
          PFSTAT(PTEFRA)=ADDR
      IF PAGSTAT=CON
        THEN PFSTAT(PTEFRA)=CON

IF VTUSCNT=0 /* VPOOL page not in use, but still connected to block */
  THEN BLKSTAT(VTBLKN)=PGCON
      BLKPAG(VTBLKN)=VPAG
      PAGSTAT=CON|DISC
      IF PAGSTAT=ADDR THEN *** ERROR ***
      IF PAGSTAT=CON
        THEN PFSTAT( )=(CON or ADDR)
          * PFSTAT=CON if page-out in process
          * PFSTAT=ADDR if PAGSTAT=CON due to VIOFREE, no page I/O
            in process

IF VTUSCNT<0 /* VPOOL page is free */
  THEN VTBLKN=NIL
      PAGSTAT=DISC

```

---

## Load Leveling

In regard to unnecessarily high paging activities in the system - that is thrashing - the page management provides algorithms to measure and to reduce high paging activities. This is done by the deactivation of one or more partitions/classes. Deactivation means, that no paging requests are satisfied for the partition/class; however, the partition/class is still in the dispatching queues and may be dispatched.

**Note:** Dynamic partitions are deactivated/reactivated by deactivating/reactivating the whole class. Whenever partition is mentioned in this paragraph, static partition or dynamic class is meant.

When thereafter the paging activities are dropped under an acceptable level, the deactivated partition(s) can be reactivated.

## Load Leveling Parameters

The load leveling algorithm is managed by so called load leveling constants which are determined by size and speed of the processor type.

NPI	Maximum number of page-ins during measurement interval
ACONST	Maximum number of page-ins per second
MINTIME	Minimum time interval for reactivation measurement

There are some further variables indicating actual values of the paging environment. They are listed below:

PIDCTR	No. of page-ins for deactivation measurement interval
PIRCTR	No. of page-ins for reactivation measurement interval
TIME1	Begin of reactivation interval
TIME2	Actual time at reactivation measurement
TIMEA	Begin of deactivation interval
TIMEB	Actual time at deactivation measurement
RRCTR	Reentry rate during deactivation interval
RRCTRX	Reentry rate during reactivation interval
EXPAVD	Exponential average of $NPI/(TIMEB-TIMEA)$
EXPAVE	Exponential average of $PIRCTR/(TIME2-TIME1)$
EXPAVR	Exponential average of $RRCTR/(TIMEB-TIMEA)$
EXPAVX	Exponential average of $RRCTRX/(TIME2-TIME1)$
REACTECB	ECB set up for a timer interval; after posting the reactivation can take place.

As system parameters the following variables are used by the load leveling routines:

IJBAPNO	Number of active static virtual partitions plus number of active dynamic classes
NDEACTP	Number of deactivated static partitions plus number of deactivated dynamic classes

## Considerations to the Parameters

### Exponential Average of Page-Ins per Second (for Deactivation)

The exponential average is a value which is calculated periodically (every time NPI page-ins have occurred). The old exponential average is used to calculate the new exponential average:

$$\text{New exp. av.} = \text{EXPAVD}' = (\text{EXPAVD} + (\text{NPI}/\text{measurement period}))/2$$

The measurement period is the time between the time when PIDCTR reached NPI (and was reset to zero) and the moment when it reaches this value again.

When NPI page-ins have occurred for the first time after IPL, the old exponential average does not exist. It is, therefore, set equal to  $NPI/\text{measurement period}$  and then the above formula is applied. Analogously, the exponential average EXPAVR is defined as the reentry rate RRCTR per second during the deactivation measurement interval.

## Reentry Rate

The reentry rate is equal to the number of page-ins of pages that were paged-out earlier in the same measurement period. To establish this value, a reentry rate counters RRCTR and RRCTR' are maintained. This counter is set to zero at the start of each measurement period. If the page manager determines that a page which is to be paged-in was paged-out earlier in the same measurement period, it increases the reentry rate counter by one. This procedure makes use of the reentry rate tables RTAB and RTABX, which are bit strings containing a bit for each page in the virtual storage. At the beginning of a measurement period, all bits of RTAB respectively RTABX are set to zero.

When the page manager determines that a page is to be read in from the page data set, the bits in RTAB respectively RTABX corresponding to the page is tested. If this bit is on, reentry is detected, and the reentry rate counter RRCTR respectively RRCTR' is increased by one.

## Deactivation Algorithm

After completion of a page-in request the variable PIDCTR is increased by one and tested if it is got equal to the constant NPI. If so, control is passed to the DEACT routines and further condition for deactivation are checked:

```
if RTAB(page) = ON ( page previously paged out )
then RRCTR' = RRCTR + 1
else RRCTR' = RRCTR

if RTABX(page) = ON ( page previously paged out )
then RRCTR' = RRCTR + 1
else RRCTR' = RRCTR
```

**Note:** Here and in the following formulas the new value of the variable xxxx is noted as xxxx', the original value is noted as xxxx.

```
if PIDCTR + 1 < NPI
then PIDCTR' = PIDCTR + 1
else PIDCTR' = 0
   RTAB' = 0
   RRCTR' = 0
   TIMEB' = actual time
   TIMEA' = TIMEB'
   EXPAVD' = (EXPAVD + NPI/(TIMEB'-TIMEA'))/2
   EXPAVR' = (EXPAVR+RRCTR/(TIMEB'-TIMEA'))/2
   if EXPAVD' >= DCONST
       then free page frames kept by FAST_CCW_X
   if EXPAVD' >= ACONST and 2*EXPAVR' > EXPAVD'
       then deactivate
```

If the deactivation conditions are satisfied, the virtual partition with the currently lowest dispatching priority is selected for deactivation. The set of these partitions is given by the formula:

```
( part (deactivation)) =
    (part | part = not(POWER or VTAM or ICCF or CICS or OCCF)
      & part = virtual
      & part = not(deactivated or TPIN or inactive)
      & part = not(open ACBs) )
```



```

if number (part(deactivation)) > 1
  then DEACT_P' = min_disp_priority(part(deactivation))
      REACTECB' = 4 sec
      NDEACT' = NDEACT + 1
      IJBAPNO' = IJBAPNO - 1
  else DEACT_P' = not determined
      REACTECB' = REACTECB
      NDEACT' = NDEACT
      IJBAPNO' = IJBAPNO

```

Deactivation means that no user page fault will be handled anymore. However, if the deactivated partition owns the LTA or other system resources the deactivation is delayed until the resources are released.

## Reactivation Algorithm

Whenever the dispatcher algorithm doesn't find a task ready to run the system enters into ALLBOUND state. During this cycle a load leveling routine checks the criteria for reactivation of partitions - if there are any. There are two different types of reactivation:

- the `u n c o n d i t i o n a l` and
- the `c o n d i t i o n a l` reactivation.

Unconditional reactivation is done if:

- there is no active virtual partition or
- no I/O is queued to any PUBS other than CRT or TP devices

Conditional reactivation is done if:

- exponential average of page-ins not greater than CCONST and
- measurement interval not lower than MINTIME

After completion of a page-in request the variable PIRCTR is increased by one.

```
PIRCTR' = PIRCTR + 1
```

The conditions and actions are :

```

if IJBAPNO = 0      (no active virtual partition)
  then unconditional reactivation
  else if NDEACTP = 0      (no deactivated partition)
    then      (no action)
    else if ( PUB(I/O) pending & not(CRT or TP device) &
              not(U/R device under POWER) )
      then conditional reactivation
      else unconditional reactivation

```

```

TIME2' = actual time
if TIME2'-TIME1 < MINTIME
  then if unconditional reactivation
    then reactivate highest priority partition
    else (no reactivation)
  else TIME1' = TIME2'
    EXPAVE' = (EXPAVE + PIRCTR/(TIME2'-TIME1))/2
    EXPAVX' = (EXPAVX + RRCTR/(TIME2'-TIME1))/2
    PIRCTR' = 0
    RRCTR' = 0
    RTABX' = 0
    if conditional activation
      then if 4*EXPAVX' < EXPAVE'
        then reactivate highest priority partition
        else (no reactivation)
      else reactivate highest priority partition

```

Reactivation means:

```

if reactivation
  then REACT_P' = max_disp_priority(deactivated partitions)
    DEACT_P' = not determined
    DEACTP' = DEACTP - 1
    IJBAPNO' = IJBAPNO + 1
    REACTECB' = 4 sec
  else ( no action )

```

After successful reactivation all PDS devices are set to NONEMPTY in order to continue with the possibly already queued page requests for the reactivated partition(s).

The page manager will be activated if it is not yet active and gets control in any case.

### Exponential Average of Page-Ins per Second (for Reactivation)

The exponential average of page-ins per second for reactivation is calculated for both conditional and unconditional requests. The calculation is similar to the calculation of the exponential average for deactivation:

New exp. av. =  $EXPAVE' = (EXPAVE + (PIRCTR/time\ interval))/2$

Note that two other quantities are used. PIRCTR is the page-in counter for reactivation. It is reset to zero after calculation of the new exponential average, and is increased by one each time a page-in occurs. Time interval is the elapsed time between the previous call of the reactivation routines and this call.

The highest priority partition which is deactivated is selected for reactivation. This is done by scanning STATPOWN from left to right (decreasing priorities). When the partition is found, it is reactivated. The byte for the partition in DEACTPSS is posted X'FF' (was X'00'), and the entry in the system communications region indicating the number of active virtual partitions is increased by one.

## Teleprocessing Balancing (TP Balancing)

Teleprocessing balancing is a special way of load leveling which is triggered by:

1. The TBAL command (see *z/VSE Operation*, SC33-8309)
2. The combined use of SVC X'58' (TPIN) and SVC X'59' (TPOUT)
3. The occurrence of page faults.

In a system with both teleprocessing and concurrent batch processing the teleprocessing subsystem may, at certain times, monopolize system resources in order to improve its response time. The performance of batch processing is decreased. TP balancing works via the deactivation string DEACTPSS by deactivating one or more of the batch partitions on request. SVC X'58' represents the request for TP Balancing, and is issued by the teleprocessing subsystem. After a certain amount of processing has been completed, SVC X'59' must be issued in order to reset TP balancing.

The TPBAL command allows the operator to turn this special load leveling on or off. If it is off, SVC X'58' and SVC X'59' have no effect. The same is true if there is no page traffic in the system, since a page fault may trigger deactivation. The operator may turn on TP Balancing by specifying the number of partitions in which delayed processing can be tolerated. This number is stored in the TPBAL parameter in the SYSCOM.

Only as many lowest-priority partitions as indicated by the TPBAL parameter are deactivated. The partition that issued the SVC X'58' is always protected from being deactivated.



---

# Storage Management

---

## General

The storage management part of the supervisor consists of the areas

- Static Storage Allocation
- Dynamic Storage Allocation

Static storage allocation consists of the following routines:

ALLOCATE (SVC 83 - X'53')

This routine (de)allocate and reallocate partitions (virtual and real).

SETLIMIT (SVC 84 - X'54')

This routine changes partition sizes and/or sets the PFI limits for a partition.

The ALLOCATE routine is located in the SVA(24-bit)-module IJBSSM, the SETLIMIT routine in the SVA(24-bit)-module IJBSSM1. The interface between IJBSSM/IJBSSM1 and the supervisor is established via various communication areas and control blocks, especially the Storage Management Communication Area (SMCOM) which is accessible via SYSCOM.IJBSCOM and the Storage Management Control Block (SMCB see Figure 113 on page 262) as part of the PCB.

Dynamic storage allocation provides work space for (reentrant) programs as well as a dynamic load facility.

It comprises the z/VSE services

GETVIS (SVC 61 - X'3D')

FREEVIS (SVC 62 - X'3E')

CDLOAD (SVC 65 - X'41')

CDDELETE (SVC 65 - X'41')

and the services ported from OS/390

GETMAIN (allocate virtual storage)

expands in OS/390 SVC 4, SVC 10, SVC 120 depending on the chosen format

FREEMAIN (free virtual storage)

expands in OS/390 SVC 5, SVC 10, SVC 120 depending on the chosen format

STORAGE OBTAIN (allocate virtual storage)

expands in PC X'30B'

STORAGE RELEASE (free virtual storage) expands in PC X'311'

These services have been ported to VSE with VSE/ESA 2.4.

They are available both in the OS390 emulation mode and in the VSE native environment. OS/390 services and VSE services can be mixed in any combination.

**Note:** Storage obtained by VSE services (GETVIS) can only be freed by VSE services (FREEVIS) and vice versa. The reason is the different subpool concept in OS/390 and VSE.

The OS390 requests are mapped to the GETVIS/FREEVIS interface and then processed the same way as a GETVIS/FREEVIS request (see "z/OS (OS/390) Storage Management Services" on page 287).

## Static Storage Allocation - Static Partition Support

The external interface for static partition allocation (virtual/real) are the commands ALLOC/ALLOC S/ALLOC R. These commands invoke the ALLOCATE macro which expands into SVC X'53'.

There are 12 static spaces supported (additionally to the real ('R ') space and the shared area ('S ')), so that each static partition can be allocated within its own address space.

The space identifiers for the static spaces are '0 ', '1 ',.. '9 ', 'A ', 'B '.

For each static partition there is a corresponding default space id, which is taken if no space id has been specified in the ALLOC command.

Partition	Default Space Id
BG	0
F1	1
.	.
.	.
FA	A
FB	B

## Static Storage Allocation - Dynamic Partition Support

There is no external command interface for dynamic partition allocation. Dynamic partition allocation is requested internally by VSE/POWER with SVC X'53' when a (POWER) job is scheduled for a dynamic partition.

Allocation values as well as the permanent SIZE value are taken from the class table.

The SIZE parameter in the EXEC statement has the same meaning as for static partitions.

During (De)Allocation various supervisor services are called. These are:

- TSRALLOC  
to check the class and to reserve a dynamic partition.
- TSRALLER  
to free the reserved partition in case an error occurred during allocation.
- TSRACT  
to activate the dynamic partition.
- TSRDEALL  
to check the PIK and to deactivate the dynamic partition.

### Dynamic Partition Allocation - Control Block Handling

The control blocks for a dynamic partition are allocated in a special subpool in the system GETVIS area during allocation processing and freed during deallocation processing. The subpool has a part in the system GETVIS area (24-bit) for control blocks which must be located below 16 MB and a part in the system GETVIS area (31-bit) for control blocks which may be located anywhere in storage.

To reduce calculation of storage requirements the requirement for the fixed length control blocks (PCB, COMREG,..) is calculated during supervisor generation.

During IPL, the requirements for machine dependant and IPL dependant (for example, number of added devices) control blocks are added.

During allocation processing only the class dependant parts need to be calculated. This is done during the first allocation request for a class. The length is then stored in the class PCB. After a PLOAD request the length is cleared and must be calculated again.

Normally three GETVIS requests are done: One for fixed control blocks (24-bit) one for fixed control blocks (31-bit) and one for the non fixed ones. The fixed control blocks are located in the partition-related subpool IJBP<syslog id>, the non fixed ones in the system-related subpool IJBNFC.

Since an allocation unit for a subpool is one page it is calculated whether the non fixed part and the fixed part (31-bit) fits in the already needed pages for the fixed request (24-bit). In this case only one subpool request is done.

During deallocation processing of a partition the whole subpool IJBP<syslog id> is freed. In the subpool IJBNFC only the area reserved for the partition is freed.

**Note:** The PCB and SCB may not cross a page boundary (Page Manager dependency).

### Interface IJBSSM - VSE/POWER

POWER does not use the ALLOCATE macro.

Input: Register 1

Register 1 = x'800000'<class> (Allocation Request)

Register 1 = x'0000'< pik > (Deallocation Request)

Output: Register 15 (Return Code)  
Register 1 (only for allocation request)

Register 1 = PCEPTR (Return Code 0)  
Register 1 = x'....'<syslog id> (Return Code 28)  
Register 1 = undefined for all other return codes

The following return codes apply only for dynamic partition (de)allocation. They are completely handled by VSE/POWER (messages, etc.).

- Register 15 = 0 (De)Allocation completed successfully
- Register 15 = 8 Invalid Class (Allocation request)  
Invalid PIK (Deallocation request)

The following return codes apply only for an allocation request.

- Register 15 = 12 Class disabled for dynamic partitions
- Register 15 = 16 No free partition within class available
- Register 15 = 20 No free partition available in the system
- Register 15 = 24 Not enough virtual storage available
- Register 15 = 28 Subpool IJBPxx does already exist
- Register 15 = 32 Not enough system GETVIS storage available
- Register 15 = 36 Not enough PFI storage available
- Register 15 = 60 Not enough storage for vendor control blocks

## Static Storage Allocation - (De-)Allocation of Page Manager Tables

The following applies for non-shared partitions only, since the page manager tables for the shared areas are allocated during IPL.

The interface between IJBSSM and the supervisor is the macro SPMRSERV whereas the page manager itself is invoked by using the macro GETPMT. When the first partition within a space (virtual/real) is allocated (SCB.SCBVSTO=0), the page manager is called to allocate the tables for this space (SPMRSERV ID=ALLPMRT). They are allocated within a page manager address space. Since the page tables for the private area must be contiguous, the maximum allocation value for the private area must be determined during space creation. This value is passed to the page manager in SCB.SCBPASZ and would normally be PASIZE. To avoid a waste of storage in case of smaller partitions the following rules apply:

- static (virtual) partitions
  - SCBPASZ = PASIZE if space id is specified explicitly
  - SCBPASZ = ALLOC value if space id is omitted (defaults used)
- static (real) partitions
  - SCBPASZ = RSIZE
- dynamic partitions
  - SCBPASZ = allocation value as specified in the class table

For that reason, it is not possible to increase the initial allocation value of a partition that was allocated by using defaults.

*Exception:*

For space '0', page tables are allocated for PASIZE. Therefore it is possible to increase the size of the BG partition even it is a default space.



When the last partition within a space is deallocated, the page manager tables are freed by using the macro `SPMRSERV ID=FREPMRT`. They are not freed when the allocation value decreases but the space still exists.

### **Static Storage Allocation - Size Processing**

With the `SIZE` command, the area within the partition, that is available for program execution, is changed. This change is permanent. The permanent start address of the `GETVIS` area (`SMCB.SMVGVIS`) is set. The command is available for static partitions only. For dynamic partitions the permanent `SIZE` value is taken from the class table.

The `SIZE` parameter of the `EXEC` statement has the same function but the change is temporary (only for the current job step). The temporary end of the program area is set (`COMREG.PPEND`). Temporary start of the `GETVIS` area is `PPEND + 1`. Both the `SIZE` command and the `SIZE` operand invoke the `SETLIMIT` macro which expands into `SVC X'54'`.

A `SIZE` value, that does not leave the minimum `GETVIS` area of 48KB below 16MB is rejected.

### **Static Storage Allocation - SETPFIX Processing**

With the `SETPFIX` command the `PFIX` limits for a partition are set (`partSMCB.SMAXPFIX`, `partSMCB.SMAXPFIX3`). These limits determine how many frames are available for `PFIX` processing. All frames, that are not reserved for partition `PFIX` processing are made available for system use (added to `sysSMCB.SMAXPFIX`, `sysSMCB.SMAXPFIX3`).

### **Static Storage Allocation - Partition Information**

The actual boundary between the partition and its `GETVIS` area can be found in the corresponding partition communication region at label `PPEND` (`PPEND + 1 = begin of partition GETVIS area`). All information about permanent partition boundaries can be found in the Storage Management Control Block (`SMCB`), (see Figure 113 on page 262), which is part of the Partition Control Block (`PCB`). An address table, pointed to by `SYSCOM.IJBASMCB` (`SYSCOM.ASMCB`) provides addressability to the specific `SMCB` entries.

SMCB Address Table Format:

Address of System Entry	Address of BG Entry	Address of FB Entry	••	Address of F1 Entry	Reserved for dyn. part.	••	Reserved for dyn. part.
0	4	8	12		52		52+(n-1)*4

n = NPARTS(SYS NPARTS= ) - no of static partitions (12)

SMCB Entry Format (SMCB)			
DEC	HEX	Label	Description
0	0	SMAXPFX	Partition: PFX (24-bit) limit in pages System : SVA(24-bit) PFX limit in pages
4	4	SMPFIX	Partition: PFX (24-bit) PFX count in pages System : SVA (24-bit) PFX count in pages
8	8	SMAXPFX3	Partition: PFX (31-bit) limit in pages System : SVA(31-bit) PFX limit in pages
12	C	SMPFIX3	Partition: PFX (31-bit) PFX count in pages System : SVA (31-bit) PFX count in pages
16	10	SMPSAVE	Partition: Save area address System : Reserved
20	14	SMVGVIS	Partition: permanent start of GETVIS area
24	18	SMVPBEG	Virtual Partition Begin Address
28	1C	SMVPEND	Virtual Partition End Address + 1
32	20	SMRPBEG	Real Begin Address
36	24	SMRPEND	Real End Address + 1
40	28	←	Length of SMCB

Figure 113. Format of Storage Management Control Block (SMCB) and SMCB Address Table

## Dynamic Storage Allocation

Dynamic storage allocation performs the management of the various GETVIS areas. Furthermore it includes a dynamic load facility.

## Address Space Layout and GETVIS Areas

There are 3 different GETVIS areas within an address space

- Partition GETVIS area  
The partition GETVIS area may cross the 16MB line depending on the allocation value. The partition GETVIS control information (PCI) is located at the high end of the partition.
- System GETVIS area  
There is a (24-bit) and a (31-bit) system GETVIS area, whereas the (31-bit) area may not exist or may reside partly or totally below 16MB. The system GETVIS control information (SCI) is located at the begin of the (31-bit) area for both the (31-bit) and the (24-bit) area. It is located at the begin of the (24-bit) area, if the (31-bit) area does not exist or is too small to hold the control information.
- Dynamic Space GETVIS area (only for dynamic partitions)  
There is only a (24-bit) Space GETVIS area. The control information (DCI) is located at the begin of the area. The Space GETVIS area is an extension to the System GETVIS area (with similar properties) and has been introduced to reduce System GETVIS requirements.

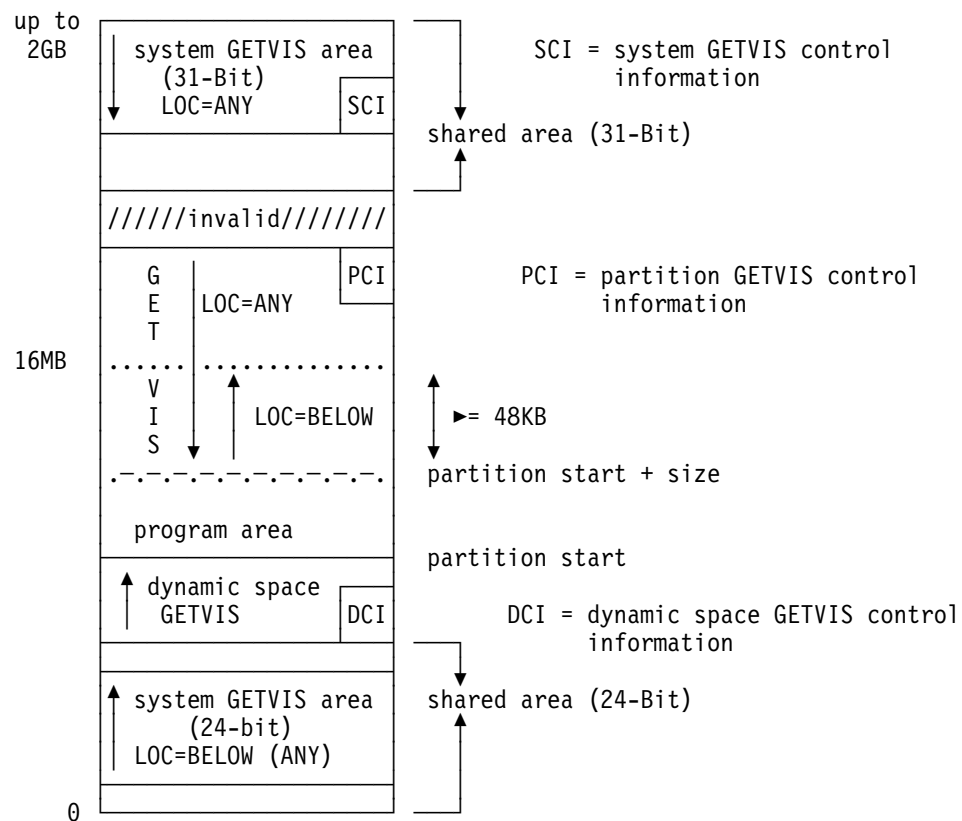


Figure 114. Dynamic Space/Partition Layout (with System GETVIS Area(31-Bit) Located Above 16MB)

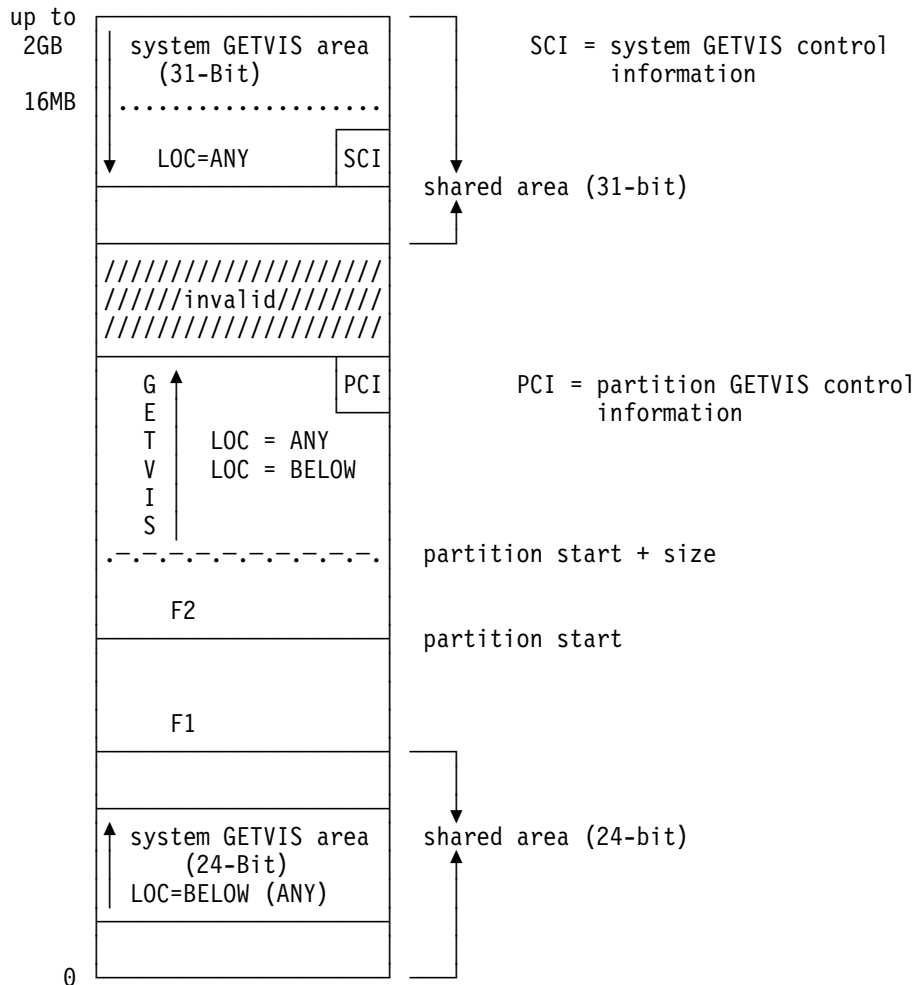


Figure 115. Static Space/Partition Layout (System GETVIS Area(31-Bit) Located Partly below 16MB)

## GETVIS Processing

From the internal point of view, a GETVIS area is divided in 2 parts:

1. Storage available for the user (from now on called GETVIS area)
2. Control information (CI), which reflects this storage and shows free/occupied areas

GETVIS/FREEVIS processing is mainly done in the CI, that means:

- GETVIS request  
It is checked whether the CI shows enough free storage to satisfy the request. If yes, the free storage is marked as occupied in the CI and the address within the GETVIS area, that is reflected by the free storage in the CI, is calculated.
- FREEVIS request  
The GETVIS area to be freed is mapped to the CI and the occupied storage in the CI is shown as free.

## Mapping between GETVIS Area and Control Information

The GETVIS area is managed on a page base, that means, it is logically divided into pages (4KB). Each page is described by a page descriptor, called subpool chain table entry (see Figure 123 on page 281). All chain table entries build the subpool chain table. They are chained in ascending order. BSUBPCHN is the begin of the subpool chain table.

Since a page is a too rough unit, each page (and therefore the GETVIS area) is sub-divided in allocation units (which is 128 byte in the partition and 16 byte in the SVA/SPACE GETVIS area). The allocation units of the GETVIS area are mapped by a bit string called VISTAB (virtual storage table) pointed to by BVISTAB. Each bit in the VISTAB represents an allocation unit (128(16) byte).

A GETVIS request can be done in multiple of allocation units (it is always rounded to the next higher multiple of an allocation unit).

### Relation between GETVIS Area, page descriptor and VISTAB

The following Figure 116 shows a GETVIS area which is divided in  $n$  pages  $P_1, \dots, P_n$ . Each page is sub-divided in  $m$  allocation units ( $m=32(4096/128)$  for partition,  $m=256(4096/16)$  for SVA/Space). Since each allocation unit is represented by a bit in the VISTAB, a page is represented by  $m$  ( $m/8$ ) VISTAB bits (bytes).

Vistab bit  $B_{xy}$  represents allocation unit  $P_{xy}$ ,  $x=1, \dots, n$ ,  $y=1, \dots, m$

$B_{xy} = 1$ : allocation unit  $P_{xy}$  is occupied

$B_{xy} = 0$ : allocation unit  $P_{xy}$  is free

The mapping between page descriptor and VISTAB is done by SPCHVSTB, which is the byte offset of the page in the VISTAB.

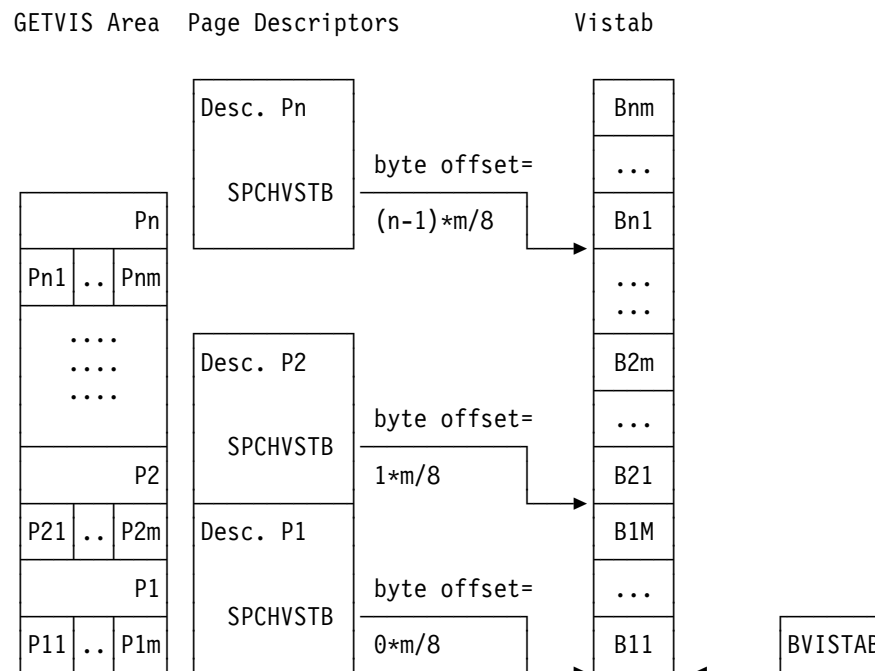


Figure 116. Relation Page Descriptor - VISTAB

## Subpool Concept

A subpool is identified by a name and consists of a number of pages belonging to that subpool. The number depends on the GETVIS/FREEVIS requests given for that subpool. A page cannot be shared between different subpools. If a page does not belong to a subpool it is enqueued to the pool of empty pages (free queue). GETVIS works on a subpool base, that means, each GETVIS request is done for a certain subpool, either specified explicitly by the user through the GETVIS SPID operand or for the general subpool 0 if the SPID operand has been omitted. Each subpool is described by an entry in the subpool index table (see Figure 124 on page 281). The entry is built when the first request for a subpool is done. The pointer to the page descriptor of the first subpool page (SPITFRST) is part of the subpool index table entry. The page descriptors of all pages belonging to a subpool are chained together by forward (SPCHFORW) and backward (SPCHBACK) pointers contained in the page descriptor. Therefore with SPITFIRST all other page descriptors of a subpool can be accessed. They are chained in ascending order.

After initialization of the GETVIS area all page descriptors are chained together in ascending order and build the subpool chain table. All pages belong to the pool of empty pages. This pool is not represented by an entry in the subpool index table but by a pointer (FIRSTPNT) to the first free page (page descriptor). After initialization this is the begin of the subpool chain table (BSUBPCHN). A page is free if all VISTAB bits representing this page are zero. The page descriptor of this page is then enqueued to the pool of empty pages. If at least one allocation unit within a page is used (corresponding VISTAB bit set), the page descriptor of this page is enqueued to a subpool.

When the first request for a subpool is done an entry is built in the subpool index table and the number of pages necessary to satisfy the request are taken from the free queue and enqueued to the subpool. Enqueue/dequeue means that the forward and backward pointers of the affected page descriptors are updated. Note, that the ascending order is always kept. As many bits as allocation units are requested, are set in the VISTAB. If the request is not a multiple of a page there is free storage within the pages enqueued for the subpool. For this storage the corresponding VISTAB bits are not set.

Figure Figure 117 on page 267 shows the subpool chain table with 3 pages enqueued to subpool S1 and one page enqueued to subpool S2. All other pages belong to the pool of empty pages (free queue).

Subpool chain table (page descriptors)

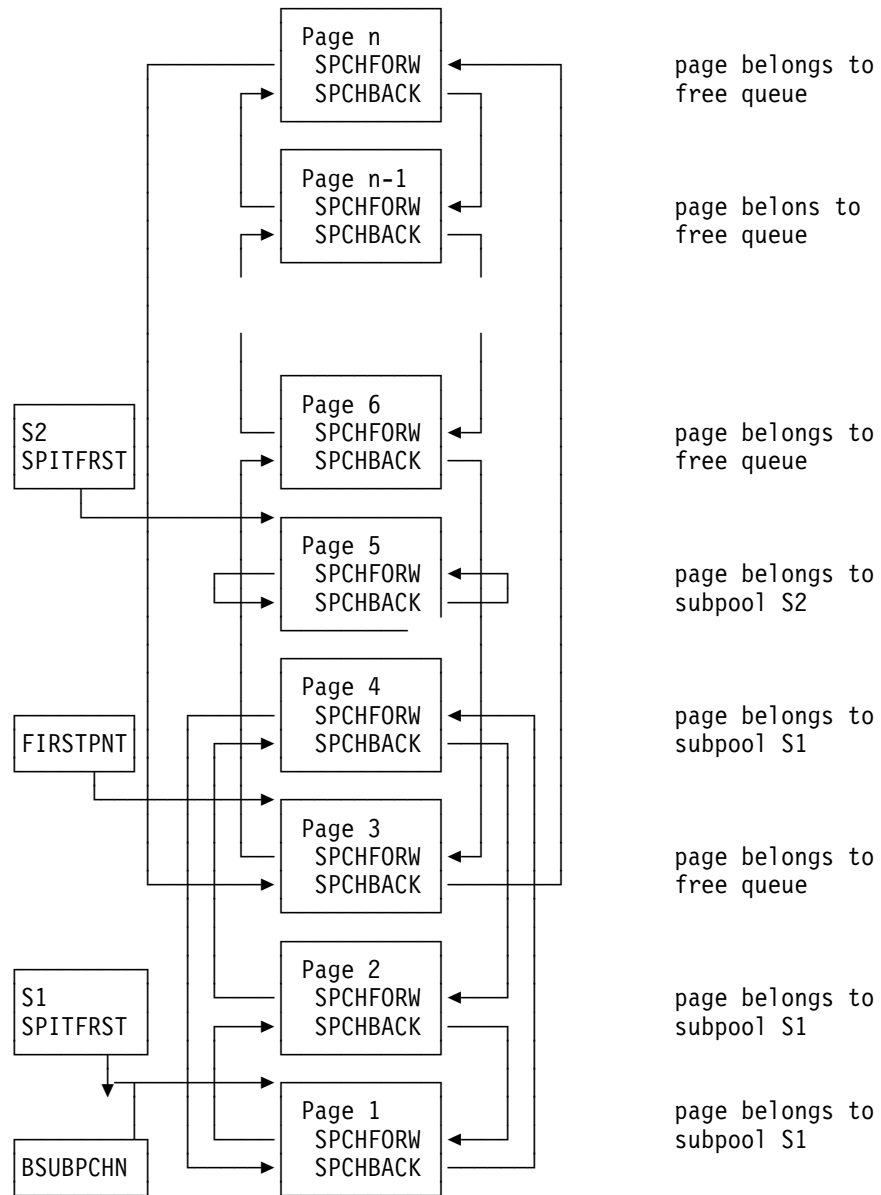


Figure 117. Subpool Concept

### Subpool properties

Within each GETVIS area there may be one or more subpools, which are managed separately. Each of the subpools has the following properties:

- A subpool may be created within the partition, system or space GETVIS area.
- The maximum number of subpools for each partition/space GETVIS area is 128. For the system GETVIS area it is  $128 + (4 * \text{number of dynamic partitions})$ .
- Each subpool consists of a number of pages which are allocated dynamically.
- All GETVIS requests which do not specify a specific subpool are satisfied within a general subpool.

- Subpool pages are only contiguous if they are requested contiguous, that means, when requesting more than one page.
- Empty pages are automatically deallocated from the subpool.
- Each task may own one subpool within a partition for exclusive use.
- Each subpool, except the general and the exclusive subpool, is defined by means of a 8-byte name which consists of a 6-byte user supplied name and a concatenated 2-byte system supplied identifier.
- All subpools, except the exclusive one, may be accessed by each task of the corresponding partition.
- Single SVA subpool pages may be PFIxed if requested by the caller.
- SVA subpools may be fetch protected (only for internal GETVIS calls).
- SPACE subpools may be fetch protected or protected with the key of the partition.
- Subpools may be created controlled, that means, subpool access is only allowed with the correct index.
- an entry in the subpool index table is deleted by a FREEVIS subpool request. It is not deleted when the last page of the subpool is freed.

## GETVIS Algorithm

### First Fit Algorithm

Since at least one allocation unit is occupied within a page belonging to a subpool, a request of at most 2 pages minus 2 allocation units can be fulfilled within an existing subpool. The GETVIS algorithm for this kind of requests is implemented as a first fit algorithm, meaning that the first gap within the subpool, that meets the requirements, is taken to satisfy the request. To avoid an unnecessary search from the begin of the subpool, a current pointer (see Figure 118 on page 269) is maintained, saying that there is no gap in the subpool before the current pointer. If the current pointer is set optimally it points to the first gap in the subpool. The search is done from the current pointer till the end of the subpool until the requested number of contiguous free VISTAB bits is found. If there is not enough free space within the subpool MIN1,MIN2 processing takes place.

### MIN1,MIN2 Processing

If the request can not be satisfied within the existing subpool, pages from the free queue must be taken. It is tried to get along with  $n-2$  ( $n-1$ ) pages from the free queue by involving the existing subpool pages, where  $n$  is the requested length rounded on the next multiple of a page ( $\text{length} = x.y \text{ pages} \Rightarrow n = x+1$ ).

This is basically how GETVIS works.

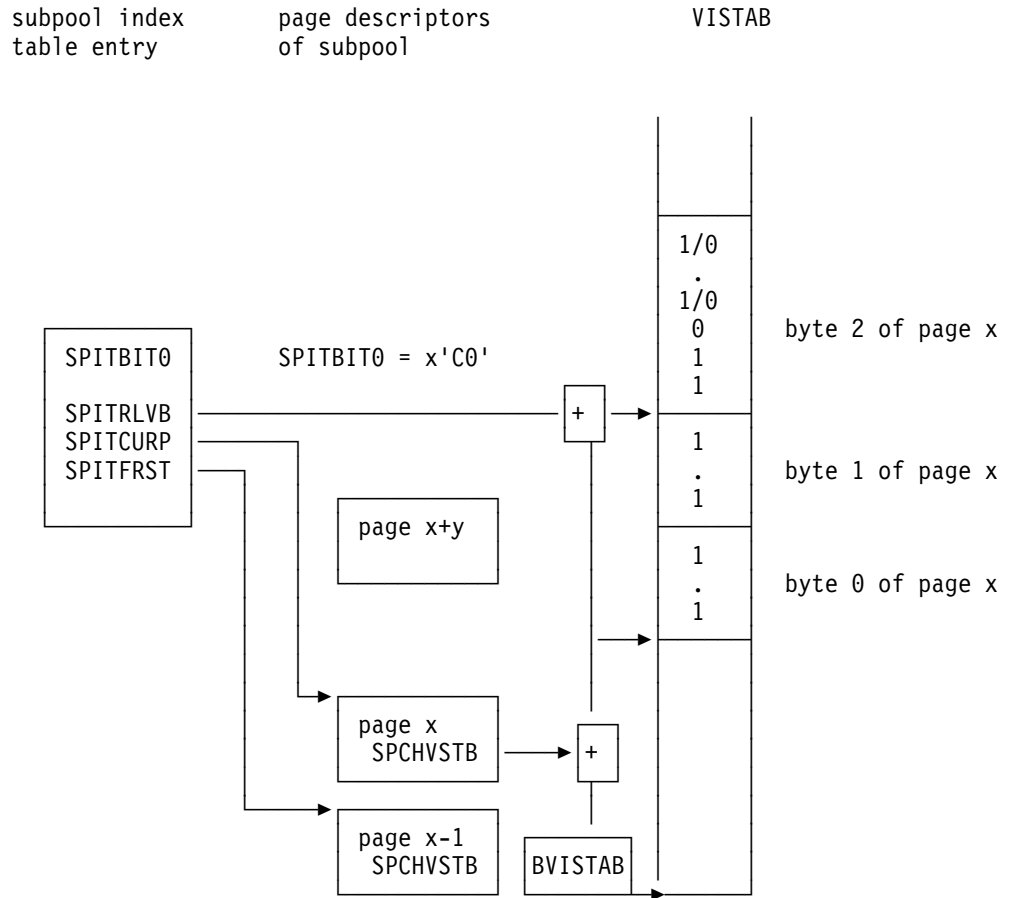
### Definition of the Current Pointer (CUR\_POINT)

The current pointer points to the first free storage in the subpool. Since a page is represented by an entry in the subpool chain table and  $m$  ( $m/8$ ) VISTAB bits (bytes) the CUR\_POINT is a triple consisting of

- SPITCURP
  - pointer to the chain table entry of the first page in the subpool with a possible gap. SPCHVSTB gives the offset of this page in the VISTAB.
- SPITRLVB
  - SPITRLVB is the first byte within the page's VISTAB representation where not all bits are set. ( $\text{SPITRLVB} = 0, \dots, m/8-1$ ).
- SPITBIT0
  - mask for bits set in byte SPITRLVB



The following figure shows a subpool whose first page is page x-1 (page 1 till page x-2 belong to other subpools/empty queue), and whose second page is page x, which has free storage.



GETVIS area

.....					
Page x-1			Page x		
	Px1 not free	Px2 not free	Px3 free		Pxm
.....					

Figure 118. Definition of the Current Pointer

## 31-Bit Addressing

With the introduction of an address space size of up to 2GB a GETVIS area may have a part below 16MB (GETVIS area 24-bit) and a part above 16MB (GETVIS area 31-bit) from now on called below and above area (see Figure 114 on page 263).

It must be distinguished between the two areas, since programs which cannot process 31-bit addresses (LOC=BELOW request) must be served from the below area, whereas programs that can handle 31-bit addresses (LOC=ANY request) are served preferred from the above area (see Figure 114 on page 263).

- the above area may not exist
  - partition is totally below 16MB
  - Space GETVIS area (24-bit area only)
- if both areas exist, they may be contiguous (partition GETVIS area) or not (system GETVIS area (24-bit) and system GETVIS area (31-bit)).

### External point of view

Allocation of GETVIS for LOC=BELOW requests is done bottom-up, following the first-fit algorithm.

Allocation of GETVIS for LOC=ANY requests is done top-down, following the first-fit algorithm.

### Internal point of view

From the internal point of view, the below and above area are considered as two independent GETVIS areas, each of one is described by some control information. Since the GETVIS algorithm works on the control information and passes the address in the GETVIS area, that maps the found location in the control information, the GETVIS algorithm works basically the same for both LOC=BELOW and LOC=ANY requests. The only change is in the address calculation. This means:

- for LOC=BELOW request, search is done in the control information that describes the below area.
- for LOC=ANY requests, search is done in the control information that describes the above area following the same algorithm as for LOC=BELOW requests. If a free area is found, the address is calculated as for below and 'mirrored' at the end of storage.

If the above area is exhausted, GETVIS searches in the control information describing the below area.

### *Exception*

If it is a partition request, boundary crossing is tried (see "Boundary Crossing (Partition Requests only)" on page 274).

### Control Information

Although the GETVIS area is artificially split in two separate areas, there is only one control information.

The control information (described by macro MAPGVCTL) contains general information valid for both the below and above area, and data describing either the below or the above area (area 2 and 3 in Figure 119 on page 271). Within the mapping structure both areas can be accessed by labels. Depending on the LOC request GETVIS works either with the data describing the below or the one describing the above area. If GETVIS would work either with area 2 or area 3, most routines would have to be duplicated using the different labels. To avoid this and to allow that both requests can be treated basically the same way there is a so-called

active area (area 2) (see Figure 119 on page 271) with which GETVIS works. Since area 2 contains normally the below data, this means, that for a LOC=ANY request the above data have to be moved from area 3 to area 2. A third area (area 1) is needed to save the below data. So the above data can be accessed by using the labels for below. There is no need to distinguish in the GETVIS code between the different requests. Of course, if a LOC=ANY request is followed by a LOC=BELOW request, area 2 has to be set up again for below. So, depending on the LOC request, the active area has to be set up. Flag GVFVFLGS.GVFVABVE shows whether the active area is set up for below or above.

GVFVFLGS.GVFVABVE=ON : area 2 contains above data

GVFVFLGS.GVFVABVE=OFF: area 2 contains below data

CI, as set up during  
initialization or  
for LOC=BELOW

CI, as set up  
for LOC=ANY

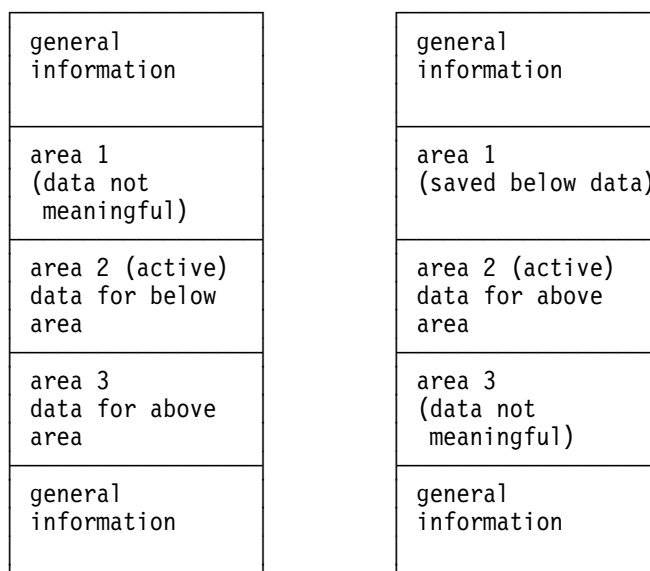


Figure 119. Control Information (MAPGVCTL) - Global Structure

A subpool can contain pages both from the below and the above area. Therefore, subpool information like first/current pointer must be available twice. Following the same principle as for the CI, the subpool CI (subpool index table entry) contains an active area, which has to be set up for below or above according to the set up of the CI. SPITFLG1.SPSWABVE shows whether the subpool is set up for below or above. The area to save the subpool fields during rotation is not part of the subpool index table entry, since it must be available only once. Note, that the same is true for the free queue, where FIRSTPNT/AFRSTPNT points to the first page descriptor within the free queue for the below/above area.

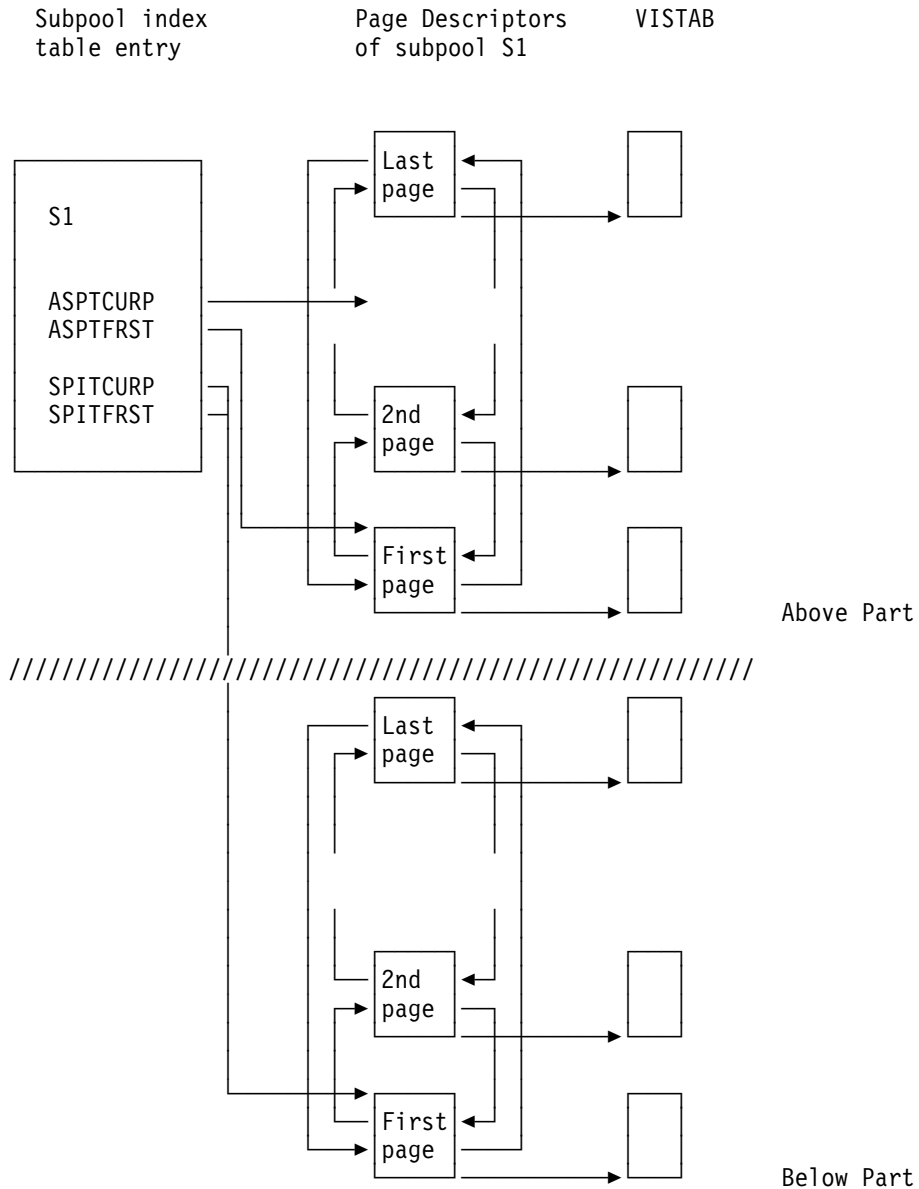


Figure 120. Relation CI - Subpool CI

### Duplicated Fields

In order to divide the GETVIS

Area into the two distinct parts, the Control Information must duplicate the following fields contained in area 2 and area 3 during initialization.

BVIRTMEM, ABVIRTMEM (begin of below/above GETVIS area)

EVIRTMEM, AEVIRTMEM (end of below/above GETVIS area)

BVISTAB, ABVISTAB (begin of VISTAB for the below/above area)

EVISTAB, AEVISTAB (end of VISTAB for the below/above area)

BSUBPCHN, ABSUBCHN (begin of page descriptor chain for below/above)

ESUBPCHN, AESUBCHN (end of page descriptor chain for below/above)

GTVSHIGH, AGTVSHI (high water mark for below/above)

FIRSTPNT,AFRSTPNT (pointer to first page descriptor in free queue below/above)  
 CURPOINT,ACURPNT (pointer to current page descriptor in free queue below/above)  
 NBRGVPG,ANBRGVPG (no of pages in below/above area)  
 GTVSPGCT,AGTVSPCT (no of used pages in below/above area)  
 GTVSMXCT,AGTVSMCT (no of pages to be used in below/above area)

Following fields are duplicated in the subpool index table entry:

SPITFRST,ASPTFRST (pointer to first page descriptor below/above)  
 (SPITCURP,SPITRLVB,SPITBIT0),(ASPTCURP,ASPTRLVB,ASPTBIT0)  
 (current pointer below/above)

**Notes:**

1. If the above area does not exist, the above fields are not set.
2. During initialization of the GETVIS area, area 2 contains the below data and area 3 the above ones.

## Location of the Control Information

**Partition GETVIS Area**

The CI, pointed to by COMREG.IJBGVCTL, is located at the high end of the partition.

If the area above 16MB holds only the CI, the above fields are not set (LOC=ANY treated as LOC=BELOW).

**System GETVIS Area**

The CI, pointed to by SMCOM.SMCSGV31, is located at the begin of the above area. If the above area holds only the CI, the above fields are not set (LOC=ANY treated as LOC=BELOW).

**Space GETVIS Area**

The CI, pointed to by PCB.PCBDYSPC, is located at the begin of the below area (above area does not exist).

## Mirroring

From the user point of view, the space in the above part is allocated top-down, although the search algorithm works bottom-up. This problem is solved by implementing a mirroring function for addresses in the above part.

**GETVIS requests**

The address, calculated by the search algorithm in the above part, is mirrored at the end of storage line (AEVRTMEM) before it is returned to the user.

ADDR : address calculated by the search algorithm

LENGTH: length of requested area

M\_ADDR: mirrored address (as passed to the user)

M\_ADDR = AEVRTMEM-(ADDR-ABVRTMEM+LENGTH)

**FREEVIS requests**

If the address passed by the caller belongs to the above part it is re-mirrored before it is fed into the algorithm.

M\_ADDR: address passed by the caller  
 LENGTH: length of area to be freed  
 ADDR : address to be fed in the algorithm  
 ADDR = AEVRTMEM-(M\_ADDR-ABVRTMEM+LENGTH)

## Boundary Crossing (Partition Requests only)

If a LOC=ANY cannot be served totally from the above area, it is tried to include the boundary in the search, that means, take storage that is adjacent to the 16MB line both from the above (as much as possible) and the below part to fulfil the request.

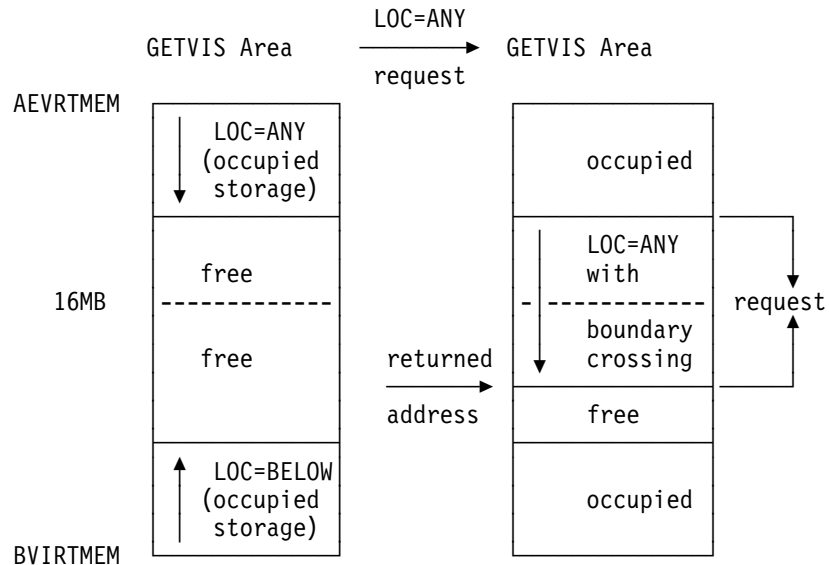


Figure 121. Boundary Crossing

## GETVIS (SVA) PFIx Processing

A GETVIS PFIx request can be divided into 3 parts:

1. Handle GETVIS request (including update of the control information)
2. PFIx GETVISed area
3. Set PFIx indication in the control information, respectively do an internal FREEVIS of the area if the PFIx request fails.

For point 1 and 3 the GETVIS gate must be closed. To avoid a deadlock, that can occur, if a task owning the GETVIS gate, is set into 'waiting for TFREE', the GETVIS gate is opened before the PFIx routine is called and closed again after the PFIx processing is done.

When the GETVIS gate is opened, further system GETVIS requests, especially to the subpool for which a PFIx is pending, must be handled:

- Before the GETVIS gate is opened and the PFIx routine is called, a PFIx pending indication is set for the subpool. If this indication is set, FREEVIS of this subpool is not allowed (RC = 'x'28')
- If the subpool is created by this request (SPITFLAG.SP CREATE set) and the PFIx request fails, an internal FREEVIS of the subpool is done.

- If the PFIX request fails and the 'subpool create' indication is not set, only the reserved GETVIS area is freed.

Flow of control of a GETVIS PFIX request:

- Gating
- Handle GETVIS request
- Open GETVIS gate
- Do PFIX storage
- Close GETVIS gate
- Open PFIX gate
- Update control information
- Do cleanup processing if PFIX failed
- Open GETVIS gate

### Gating

- Close PFIX gate and gate GETVIS (close gate or set RID).  
This is only done if both gates are free. Otherwise GETVIS waits until both gates are free. The PFIX gate is closed by the GETVIS processing and no longer by the PFIX routine. This is done for several reasons:
  - The save area for the registers destroyed by the PMR is available only once.
  - In case of a PFIX request, the subpool offset and the address of the subpool name in the users are saved. They are necessary for FREEVIS processing if the PFIX request fails. These areas are located in the GETVIS control information, therefore only one PFIX request at a time is allowed to enter the GETVIS code.
- Open GETVIS gate before calling PFIX
  - avoid deadlock
- Close GETVIS gate after PFIX  
By the same reason as described above, the GETVIS gate is closed again after PFIX processing before opening the PFIX gate.
- open PFIX gate
- open GETVIS gate

## GETVIS/FREEVIS Options

*Input for GETVIS service (SVC X'3D'):*

R0: Length of requested area

R1:

- Not required or
- Pointer to area to start search (if POOL specified) or
- Pointer to subpool name field (if SPID specified)

R15: Option in low order byte:

X'01': Page boundary requested (2K/4K boundary depending on length)

X'02': POOL specified

X'04': SVA space requested

X'08': Subpool specified

X'10': PFIX requested

X'20': Exclusive subpool wanted

X'40': Fetch protection requested

- X'80': Prevent page boundary crossing (only for internal calls)
- R15: Option in byte 2:
  - X'01': Excessive requestor (only internal call)
  - X'02': SPACE request
  - X'04': SPACE request with partition key protection
  - X'08': Controlled subpool
  - X'10': LOC=BELOW request
  - X'20': LOC=ANY request
  - X'40': request for pmr space (only for internal calls)
  - X'80': IPL request (only for internal use)
- R15: Option in byte 3:
  - X'01': STATUS=SIZE request (only internal call)
  - X'02': STATUS=ALL request (only internal call)
  - X'04': PREFIX=YES request (only internal call, 8K prefix area)

*Output for GETVIS service (SVC X'3D'):*

- R1: Pointer to found area
- R15: Return code in low order byte:
  - See GETVIS Macro description

*Input for FREEVIS service (SVC X'3E'):*

- R0: Length of area to be freed
- R1:
  - Pointer to area to be freed or
  - Pointer to subpool name field (if SPID specified)
- R15: Option in byte two and three
  - X'0002': Subpool specified
  - X'0004': SVA space to be freed
  - X'0008': FREEVIS ALL specified
    - EOJ: Invalidate the corresp. partition GETVIS area.
    - EOT: Free the task related exclusive subpool.
  - X'0010': ROUTED space to be freed (only for internal call)
    - called during UNBATCH processing
  - X'0200': SPACE request
  - X'4000': PMR space request

*Output for FREEVIS service (SVC X'3E'):*

- R15: Return code in low order byte:
  - See FREEVIS Macro description

## I/F GETVIS - IPL

Before allocating the SVA, IPL calls Getvis to calculate the size of the control information. At this point in time, IPL knows all system and user requirements for the system Getvis area. IPL passes this information to Getvis. Getvis returns the size of the control information.

IPL then increases the size of the 31-bit system Getvis area by this value.

Getvis initializes the control information with the first Getvis request. However, the 24-bit system Getvis area may not yet be usable, because the 31-bit system Getvis area is validated some time before the 24-bit system Getvis area. As long as the 24-bit system Getvis area is not validated, only LOC=ANY requests can be satisfied. When bit SMCGVBEL in SMCOM.SMCFLG1 is set, both areas are valid.



When Getvis is called by IPL to calculate the size of CI

- R0 contains the number of dynamic partitions
- R1 contains the size of system getvis area (24-bit and 31-bit) in bytes.

On return to IPL

- R0 contains the size of the control information in bytes.
- RF contains the return code (always 0)

# GETVIS Control Blocks

## MAPGVCTL

Figure 122 (Page 1 of 3). Layout of GETVIS Area Control Information (MAPGVCTL)

Offsets					
Dec	Hex	Type	Len	Name (Dim)	Description
0	(0)	SIGNED	4	ANCHDIR (0)	ANCHOR TABLE
				MOVECIL	"52" LENGTH FOR MOVING THE DATA
				MOVESPTL	"10" LENGTH FOR MOVING THE DATA
				OFVRTMEM	"16" OFFSET OF BVRTMEM
1024	(400)	SIGNED	4	BSUBPIND	BEGIN OF SUBPOOL INDEX TABLE
1028	(404)	SIGNED	4	ESUBPIND	END OF SUBPOOL INDEX TABLE
1032	(408)	SIGNED	4	EGVCTLB	LAST BYTE OF CONTROL INFORM.
1036	(40C)	SIGNED	4	ENDGVCTL	END OF CONTROL AREA
1040	(410)	BITSTRING	1	DUPCILSV (0)	SAVE AREA FOR DUPLICATED FLD
1056	(420)	SIGNED	4	BBVRTMEM	
1060	(424)	SIGNED	4	BEVRTMEM	
1092	(444)	SIGNED	4	NBRGVPG	NBR OF AVAILABLE PAGES
1096	(448)	SIGNED	4	GTVSPGCT	NBR OF CURRENT USED PAGES
1100	(44C)	SIGNED	4	GTVMXCT	MAX NBR OF PAGES TO BE USED
1104	(450)	SIGNED	4	GTVSEXCT	MAX NBR OF PAGES TO BE USED BY EXCESSIVE SUBPOOLS
1108	(454)	SIGNED	4	BVRTMEM	BEGIN OF GETVIS AREA
1112	(458)	SIGNED	4	EVRTMEM	END OF GETVIS AREA
1116	(45C)	SIGNED	4	BVISTAB	BEGIN OF VISTAB
1120	(460)	SIGNED	4	EVISTAB	END OF VISTAB
1124	(464)	SIGNED	4	BSUBPCHN	BEGIN OF SUBPOOL CHAIN TAB.
1128	(468)	SIGNED	4	ESUBPCHN	END OF SUBPOOL CHAIN TABLE
1132	(46C)	SIGNED	4	GTVSHIGH	PAGE CHAIN HIGH WATER MARK
1136	(470)	SIGNED	4	FIRSTPNT	FIRST PAGE WITHIN EMPTY POOL
1140	(474)	SIGNED	4	CURPOINT	START SEARCH ADDRESS
THE FOLLOWING FIELDS ARE DUPLICATED FOR THE ABOVE PART OF THE GETVIS AREA (UP TO MOVECIL)					
1144	(478)	SIGNED	4	ANBRGVPG	NBR OF AVAILABLE PAGES
1148	(47C)	SIGNED	4	AGTVSPCT	NBR OF CURRENT USED PAGES
1152	(480)	SIGNED	4	AGTVMCT	MAX NBR OF PAGES TO BE USED
1156	(484)	SIGNED	4	AGTVSXCT	MAX NBR OF PAGES TO BE USED BY EXCESSIVE SUBPOOLS
1160	(488)	SIGNED	4	ABVRTMEM	BEGIN OF GETVIS AREA
1164	(48C)	SIGNED	4	AEVRTMEM	END OF GETVIS AREA
1168	(490)	SIGNED	4	ABVISTAB	BEGIN OF VISTAB
1172	(494)	SIGNED	4	AEVISTAB	END OF VISTAB
1176	(498)	SIGNED	4	ABSUBCHN	BEGIN OF SUBPOOL CHAIN TAB.
1180	(49C)	SIGNED	4	AESUBCHN	END OF SUBPOOL CHAIN TABLE
1184	(4A0)	SIGNED	4	AGTVSHI	PAGE CHAIN HIGH WATER MARK
1188	(4A4)	SIGNED	4	AFRSTPNT	FIRST PAGE WITHIN EMPTY POOL

Figure 122 (Page 2 of 3). Layout of GETVIS Area Control Information (MAPGVCTL)

Offsets					
Dec	Hex	Type	Len	Name (Dim)	Description
1192	(4A8)	SIGNED	4	ACURPNT	START SEARCH ADDRESS
1196	(4AC)	SIGNED	4	ADDRSAV	SAVE AREA FOR USER ADDRESS
1200	(4B0)	SIGNED	4	LGTHSAV	SAVE AREA FOR REQUEST LENGTH
1204	(4B4)	SIGNED	4	SPIDSAV (0)	ABS. ADDR(SUBPOOL NA.) IN CI
1208	(4B8)	SIGNED	4	SSEARCH	NEW START SEARCH ADDRESS
1212	(4BC)	SIGNED	4	SVWORK1	SAVE WORK REG 1
1216	(4C0)	SIGNED	4		RESERVED
1220	(4C4)	SIGNED	4	SSPPTRCI	SAVE PTR TO SP IN BSUBPIND
1224	(4C8)	SIGNED	4	SSPNPTR	SAVE PTR TO USERS SUBPOOL N.
1228	(4CC)	SIGNED	4	SAVR2ND (0)	SECOND SAVE AREA FOR SUBR
1228	(4CC)	SIGNED	4	SAVR2ND6	SAVE AREA FOR REG6
1232	(4D0)	SIGNED	4	SAVR2ND7	SAVE AREA FOR REG7
1236	(4D4)	SIGNED	4	SAVR2NDF	SAVE AREA FOR REGF
1240	(4D8)	SIGNED	4	SAVR3RD (0)	SECOND SAVE AREA FOR SUBR
1240	(4D8)	SIGNED	4	SAVR3RD7	SAVE AREA FOR REG7
1244	(4DC)	SIGNED	4	SAVR3RD8	SAVE AREA FOR REG8
1248	(4E0)	SIGNED	4	SAVR3RDF	SAVE AREA FOR REGF
1252	(4E4)	SIGNED	4	SAVREGS (16)	REG. SAVE AREA F. SUBROUTS
1316	(524)	SIGNED	4	SVPFSPID	SAVE USER'S SUBPOOL ADDR
1320	(528)	SIGNED	4	SVPFR1	SAVE USER'S REGISTER 1
1324	(52C)	SIGNED	4	SVPFLIST (2)	PFIX/PFREE PARM LIST
1332	(534)	BITSTRING	1	SVPFEND	PFIX/PFREE PARM LIST END
1333	(535)	BITSTRING	1	GVFVFLGS	NEW FLAG BYTE
		.... ..1		GVMIN2	"X'01" INDICATE MIN_2 PROCESSING
		.... ..1.		GVFVABVE	"X'02" INDICATE PROCESSING ABOVE
		.... ..1..		GVCIMOV	"X'04" CI MOVED TO BELOW PART
		.... 1...		GVFVINT	"X'08" INTERNAL FREEVIS PROCESSING
		...1 ....		GVUPDCP	"X'10" UPDATE CURRENT POINTER
		..1. ....		GVFIRSTT	"X'20" SUBPOOL'S FIRST CONT. AREA SCANNED.
		..1. ....		GVCHKPRO	"X'40" PAGES ENQUEUED, CHECK PROTECTION.
		1... ....		GVLNG0RQ	"X'80" REMEMBER LENGTH 0 REQUEST
1334	(536)	SIGNED	2	MXSUBPLH	MAX NBR OF SUBPOOLS AVAIL.
1336	(538)	BITSTRING	1	GVFVFLG2	FLAG BYTE
1337	(539)	BITSTRING	3		RESERVED
1340	(53C)	SIGNED	4	SCURPSAV	SAVE AREA FOR SPITCURP
1344	(540)	BITSTRING	1	SRLVBSAV	SAVE AREA FOR SPITRLVB
1345	(541)	BITSTRING	1	SBITOSAV	SAVE AREA FOR SPITBITO
1346	(542)	BITSTRING	A	DUPSBSV	SAVE AREA FOR DUPLICATED FLD
1356	(54C)	SIGNED	4	FPAGEPRO	FIRST PAGE TO BE PROTECTED
1360	(550)	SIGNED	4	LPAGEPRO	LAST PAGE TO BE PROTECTED
1364	(554)	SIGNED	4	GVFVGATE	ADDRESS OF AREA GATE
1368	(558)	SIGNED	8	GMFMSPNM	OS/390 SUBPOOL NAME

Figure 122 (Page 3 of 3). Layout of GETVIS Area Control Information (MAPGVCTL)

Offsets					
Dec	Hex	Type	Len	Name (Dim)	Description
1376	(560)	SIGNED	4	MAXGAP	NO OF CONT.PAGES BELOW
1380	(564)	SIGNED	4	MAXGAP_ABV	NO OF CONT.PAGES ABOVE
1384	(568)	SIGNED	4	MAXGAP_BDY	NO OF CONT.PAGES ACROSS BDY
1388	(56C)	SIGNED	4	SMAXGAP	SAVED ..
1392	(570)	SIGNED	4	SMAXGAP_ABV	MAXGAP ..
1396	(574)	SIGNED	4	SMAXGAP_BDY	VALUES
1400	(578)	SIGNED	4	GVFVGATE_SAV	SAVED DURING PFI
1404	(57C)	SIGNED	4	GMFMSPLE	
1408	(580)	SIGNED	4	SPITMVSP	ADDR OF CURRENT SPLE
1412	(584)	BITSTRING	1	SAVSHIFT	
1413	(585)	BITSTRING	1	SAVRQST	
1414	(586)	SIGNED	2	SPIDSAVH	
1416	(588)		64		SAVE AREA
1480	(5C8)	SIGNED	4	GMFMWRK1_EXT	
1484	(5CC)	SIGNED	20C	GMFMWRK2	IJBSSM2 WORK AREA
2008	(7D8)			VISTAB	*** BEGIN OF BIT PATTERN

**Note:** Due to compatibility reasons, the VSAM control information remains at the same location within the GETVIS area, that means, it has the same offsets relative to PPEND as in former releases. The mapping macro for the VSAM control information is still MAPANCH and contains only this information.

## Subpool Chain Table Entry

*Figure 123. Layout of Subpool Chain Table Entry (SUBPCHN)*

Offsets					
Dec	Hex	Type	Len	Name (Dim)	Description
0	(0)	SIGNED	4	SPCHFORW	SUBPOOL FORWARD POINTER
4	(4)	SIGNED	4	SPCHBACK	SUBPOOL BACKWARD POINTER
8	(8)	SIGNED	4	SPCHVSTB	RELATIVE PAGE VISTAB PTR
12	(C)	SIGNED	2	SPCHNMBR	SUBPOOL ID (NUMBER)
14	(E)	BITSTRING	1	SPCHFLAG	SUBPOOL PAGE FLAGS
		.... ...1		SPPGCONC	"X'01" CONCATENATION FLAG
		...1 ....		SPPGPFIX	"X'10" PAGE IS PFIXED (ONLY SVA)
15	(F)	BITSTRING	1		RESERVED FOR FUTURE USE

## Subpool Index Table Entry

*Figure 124 (Page 1 of 2). Subpool Index Table Entry (SUBPINT)*

Offsets					
Dec	Hex	Type	Len	Name (Dim)	Description
0	(0)	CHAR- ACTER	6	SPITNAME	SUBPOOL NAME FIELD
6	(6)	BITSTRING	2	SPITNMBR	SUBPOOL NUMBER
8	(8)	BITSTRING	1	SPITKEY	KEY FOR PAGE PROTECTION
9	(9)	BITSTRING	1	SPITFLAG	SUBPOOL FLAG
		.... ...1		SPFTCHPR	"X'01" SUBPOOL IS FETCH PROTECTED
		.... 1...		SPPKEYPR	"X'08" SUBPOOL IS PROTECTED WITH PARTITION KEY
		...1 ....		SPCNTRLD	"X'10" SUBPOOL IS CONTROLLED
		..1. ....		SPCREATE	"X'20" SUBP. CREATED BY CURR. REQ
		.1.. ....		SPPFXPND	"X'40" SUBPOOL WITH PENDING PFIX
10	(A)	SIGNED	1	SPITPIK	PIK OF OWNING PARTITION
12	(C)	BITSTRING	1		RESERVED
13	(D)	BITSTRING	1	SPITFLG1	SUBPOOL FLAG BYTE 2
		1... ....		SPFPEMPTY	"X'80" FIRST PAGE EMPTY
		.1.. ....		SPLPEMPTY	"X'40" LAST PAGE EMPTY
		..1. ....		SPCLPMR	"X'20" CALL PAGE MANAGER
		...1 ....		SPBTHPRT	"X'10" BOTH PARTS FREED
		.... ...1		SPSWABVE	"X'01" SUBPOOL SWITCHED TO ABOVE
14	(E)	SIGNED	2	SPITTASK	TASK THAT CREATED SUBPOOL
16	(10)	SIGNED	4	SPITPUSC	PAGE USAGE COUNT
20	(14)	SIGNED	4	SPITFIRST	PTR TO FIRST CHAIN TABLE ENTRY OF SUBPOOL
24	(18)	SIGNED	4	SPITCURP	PTR TO CURRENT CHAIN TABLE ENTRY
28	(1C)	BITSTRING	1	SPITRLVB	REL. CURRENT PTR WITHIN CURRENT PAGE
29	(1D)	BITSTRING	1	SPITBITO	OR MASK FOR SPITRLVB (IMPLICIT CURRENT BIT PTR)
30	(1E)	SIGNED	2		RESERVED

THE FOLLOWING FIELDS ARE DUPLICATED FOR THE ABOVE PART OF THE GETVIS AREA (UP TO MOVESPTL)

Figure 124 (Page 2 of 2). Subpool Index Table Entry (SUBPINT)

Offsets					
Dec	Hex	Type	Len	Name (Dim)	Description
32	(20)	SIGNED	4	ASPTFRST	PTR TO FIRST CHAIN TABLE ENTRY OF SUBPOOL
36	(24)	SIGNED	4	ASPTCURP	PTR TO CURRENT CHAIN TABLE ENTRY
40	(28)	BITSTRING	1	ASPTRLVB	REL. CURRENT PTR WITHIN CURRENT PAGE
41	(29)	BITSTRING	1	ASPTBITO	OR MASK FOR SPITRLVB (IMPLICIT CURRENT BIT PTR)
42	(2A)	SIGNED	2		RESERVED
				SUBPINTL	"*-SUBPINT" LENGTH OF SUBPOOL INDEX TAB.

## CDLOAD Support (SVC X'41')

This function loads a phase dynamically into the partition GETVIS area when called by the macro CDLOAD.

Exception: The phase is found in the SVA and the requesting program is not running in real mode (real mode checked by Fetch/Load processing).

For each phase, that is loaded into the partition GETVIS area, and entry in the anchor table, which is part of the GETVIS control information, is built.

Before the SVC X'41' routine is invoked, the name of the phase to be loaded (specified by the first operand of the CDLOAD macro) must be pointed to by general register 1.

CDLOAD first checks to see if the GETVIS area control table is already initialized; if so, the anchor table is searched for an entry for the requested phase. If an entry is found, the return parameters are retrieved from the entry and control is returned to the caller.

If the anchor table does not exist or does not have an entry for the requested phase, a LOAD is issued with the parameters DE=YES and TXT=NO. The FETCH routine moves only the directory entry for the requested phase into an area specified by CDLOAD (an area at DFWKNAME in the TCB). The CDLOAD routine then checks the directory entry: if the phase is not found, control is passed to ERR22, or the return code is passed. If the phase resides in the SVA, the required parameters are retrieved from the directory entry and passed in registers 0, 1, and 14. In addition, return code X'00' (successful completion) is passed in register 15.

A phase residing in the SVA is not added to the anchor table. If the requesting task runs in a real partition, a SVA phase is loaded into the corresponding real partition GETVIS area.

The phase name is inserted in the first free entry in the anchor table (see Figure 126 on page 285 and Figure 125 on page 285). If there is no free entry, storage for a new anchor table is obtained in the dynamic space GETVIS (system GETVIS) area. If there is no free space for a new anchor table return code X'10' is passed. SVC X'41' then obtains the length of the phase to be loaded from the directory entry and passes this information to the GETVIS routine. Depending on the RMODE of the phase a LOC=ANY respectively a LOC=BELOW request is done.

The GETVIS routine reserves the required storage and returns the load address of the phase to SVC X'41'. SVC X'41' then loads the phase by issuing a LOAD with the parameters TXT=YES and DE=YES. After completion of the load operation, the load point, the entry point, the length and the attributes of the phase are stored in the anchor table and the load count of the phase is incremented by one. If the load count is at the maximum (X'FFFF') it is not increased. Instead an indication is set to prevent a CDDELETE of the phase.

Successful completion is indicated by passing the return code X'00' in register 15. The layout of the anchor table is shown in Figure 126 on page 285. The layout of an anchor table entry is shown in Figure 125 on page 285.

*Input for CDLOAD service (SVC X'41'):*

R1: Pointer to phase name

R15: Option in low order byte:  
 X'01': Page boundary requested (2K/4K depending on length)  
 X'04': Consider only SVA phases  
       If the phase is not in the SVA, it is not loaded.  
 X'10': Return if phase not found

*Output for CDLOAD service (SVC X'41'):*

R0: Load address of phase  
 R1: Entry point of phase  
       CDLOAD set the high-order bit in register 1 to indicate the phase's AMODE  
       (0 for AMODE 24, 1 for AMODE 31). If the phase's AMODE is ANY the high-  
       order bit is set corresponding to the caller's AMODE.  
 R14: Length of phase  
 R15: Return code in low order byte:  
       • See CDLOAD Macro description

## CDDELETE Support (SVC X'41')

This function deletes a phase previously loaded by a CDLOAD request.

Since both the CDLOAD and the CDDELETE macro expand into SVC X'41' an option in register 15 indicates which function is required.

On entry to the SVC X'41' routine, the name of the phase to be deleted must be pointed to by general register 1.

If the GETVIS control information is not initialized or if there is no entry for the requested phase a return code is passed.

If the load count was exceeded by a previous CDLOAD request (ATPHFLAG.ATLOADCE set), the phase is not deleted but a return code is passed. Otherwise the load count is decremented by one. If the load count is zero, the entry in the anchor table is cleared and the storage occupied by the phase is freed (by use of the FREEVIS routine). Successful completion is indicated by return code 0.

The anchor table is freed after the last entry is cleared. Only the first anchor table which is part of the GETVIS control information is never freed.

*Input for CDDELETE service (SVC X'41'):*

R1: Pointer to phase name  
 R15: Option in low order byte:  
       X'02': CDDELETE request

*Output for CDDELETE service (SVC X'41'):*

R15: Return code in low order byte:  
       • See CDDELETE Macro description

**Anchor Table Handling** Each anchor table consists of header information followed by storage to hold the phase entries. The first anchor table is located at the begin of the partition GETVIS control information which is pointed to by COMREG.IJBGVCTL. Whenever a phase is loaded, an entry in the anchor table is built and the use count (ANCHNOUE) is incremented by one. If the anchor table is full, a new one is allocated by using the SGETVIS SPACE function. The forward (ANCHFWP) and backward (ANCHBWP) pointer is updated. The backward pointer of the first and the forward pointer of the last anchor table is set to zero. Whenever the load count of a phase is zero the entry in the anchor table is cleared and the use count of the anchor table (ANCHNOUE) is decremented by one. If the anchor



table is empty, it is freed by means of the SFREEVIS SPACE function. The forward and backward pointer of the previous and following anchor table is update accordingly.

Anchor Table Entry Layout (ATENTRY)			
DEC	HEX	Label	Description
0	0	ATPHSNME	Phase Name Field
8	8	ATLOADP	Load Point in GETVIS Area
12	C	ATENTP	Entry Point in GETVIS Area
16	10	ATPHSLEN	Length of loaded Phase
20	14	ATLDCNT	No of CDLOAD requests (maximum is X'FFFF')
22	16	ATPHATT	Flag moved from TCB.DFWKEMVS (AMODE,RMODE)
23	17	ATPHFLAG	Flag byte
		ATLOADCE	X'80' phase load count exceeded
24	18	ATLDSYS	No of system load requests
26	1A	ATPHFLG2	Flag byte
27	1B	ATSUBPOL	Subpool id of GETMAIN
28	1C	← Length of Anchor Table Entry (ATENTRY)	

Figure 125. Format of Anchor Table Entry

Anchor Table Layout			
DEC	HEX	Label	Description
0	0	ANCHFWP	Ptr to next anchor table (0 if not existing)
4	4	ANCHBWP	Ptr to previous anchor table(0 if not exist.)
8	8	ANCHNOUE	Number of used entries
10	A	ANCHNUME	Number of total directory entries
12	C	ANCHDIRF	First phase entry (described by ATENTRY)
40	28		Second phase entry
..	..		...
..	..		...
992	3E0		Last phase entry
1020	3FC		Reserved
1024	400	← Length of Anchor Table	

Figure 126. Layout of an Anchor Table

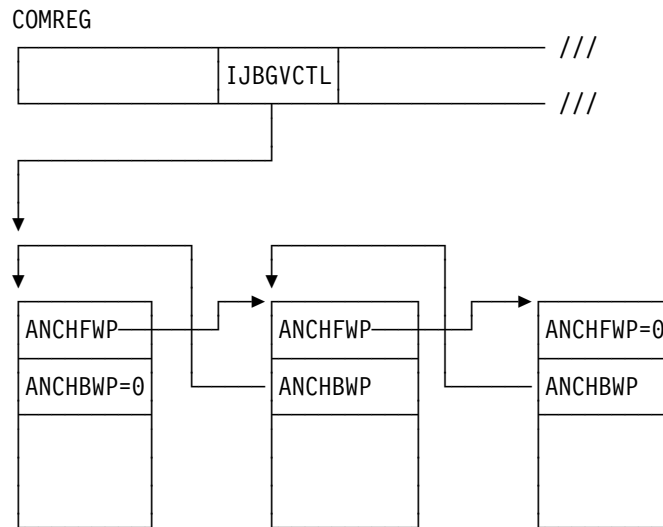


Figure 127. Example of Three Anchor Tables Allocated

---

## z/OS (OS/390) Storage Management Services

### Overview

- OS/390 requests are mapped directly to VSE requests and are then processed the same way as VSE requests. Hence, the GETMAIN/FREEMAIN/STORAGE requests and GETVIS/FREEVIS requests can be used in any combination within an application.
- Since the subpool concept of OS/390 differs from that of VSE, OS/390 subpools can only be addressed by OS/390 requests and vice versa.
- On OS/390, the allocation unit is 8 bytes.  
On VSE, the allocation unit is the same for OS/390 requests and VSE requests.
  - 16 bytes for system/space getvis
  - 128 bytes for partition getvis
- FREEMAIN and STORAGE RELEASE will always clear the storage
- Storage areas in OS/390 are the CSA, the SQA, the LSQA, and the PVT. These areas are mapped to VSE's storage areas.
- In a VSE partition, there is one Job Step Task per job, the maintask, i.e. the main task is the related job step task for all of its subtasks.

Implementation is done by new code in the macros SGAM and SGAMSUBR and in a new module, IJBSSM2(phase name = \$IJBSSM2).

- dependencies on GETVIS/FREEVIS are put into SGAM(SUBR),
- dependencies on GETMAIN/FREEMAIN/STORAGE are put into IJBSSM2. The mapping of an OS/390 request into a VSE request is mainly done in module IJBSSM2.

### Mapping of OS/390 Storage Areas to VSE

- SQA/ESQA, CSA/ECSA  
system queue area, common service area  
-> mapped to system Getvis area
- LSQA/ELSQA  
local system queue area  
-> mapped to space Getvis area
- PVT  
-> mapped to partition Getvis area  
On OS/390 the PVT is divided into a private low and private high area. This distinction is not done in VSE.

### OS/390 Subpool Concept

- In OS/390, subpools are identified by a number and not by a name. The range is 0-255. Not all numbers are used by OS/390 and only a subset is supported by VSE. Valid numbers in VSE are
  - 0-127,129-132,226-231,239-241,245,250-255
- the subpool number defines attributes which are associated with the subpool's pages such as

- location (storage area)
  - fetch protection
  - type (pageable|fixed)
  - owner (lifetime)
  - storage key
- the subpool number defines whether it's an authorized subpool. These subpools can be used by authorized programs only. A program is authorized when it runs in supervisor state, key 0 or is a CICS or vendor subsystem.

### **Subpool Owner (Lifetime)**

A subpool is created during the first GETMAIN/STORAGE request for the subpool. The subpool owner defines the lifetime of the subpool. Owner of a subpool can be a task, the maintask (job step task in OS/390), the address space or the system.

- task related storage
 

The storage is owned by a task and anchored to the task's TCB. Normally it's the task, that issued the request.

An exception is the TCBADDR parameter. The storage is owned by the TCB specified in the TCBADDR parameter.

Task related storage can be in the partition or space getvis area.

A subpool which is task related is discarded during task termination. During task termination, FREEVIS ALL is called. FREEVIS ALL will loop through all subpools owned by the task and do

  - discard the control block which describe the OS/390 subpool (SPLE)
  - freevis all subpools belonging to the task
  - clear the chain pointer in the VSE TCB
- maintask (job step) related storage
 

The storage is owned by the maintask. It is anchored to the maintask's TCB, independent which task in the partition issues the request. A job step related subpool exists only once in the partition. The storage can be in the partition or space getvis area. The subpool is discarded during maintask termination. Cleanup is done the same way as for subtasks, except that partition getvis subpools are not freed explicitly, since the partition getvis area is invalidated.
- address space related storage
 

The storage is owned by the address space and anchored to the SCB. The storage is in the space getvis area. The subpool is discarded during address space termination. Address space termination means UNBATCH for static partitions and de-allocation for dynamic partitions. During UNBATCH processing for static partitions FREEVIS ROUTED will loop through all subpools owned by the address space and do

  - discard the control block which describe the OS/390 subpool (SPLE)
  - freevis all LSQA subpools (space getvis area) subpools
  - clear the chain pointer in the SCB.

For dynamic partitions the SPLE, the SCB as well as the whole Space Getvis Area are freed during de-allocation.
- system related storage
 

The storage is owned by the system, not by a task. The subpool is only discarded when explicitly specified. The storage is in the system getvis area.

## Sharability

- System related subpools are shared by all tasks of the system
- Address space related subpools are shared by all tasks of the address space
- Maintask related subpools are shared between the maintask and all subtasks of the partition.
- Task related subpools cannot be shared.  
Exception: Subpool 0 is shared by the maintask and all subtasks.

## Storage Key/Protection

Subpools can be store and fetch protected with any key in any area. A subpool with multiple keys will be implemented as multiple subpools.

For some subpools the storage key is selectable. All other subpools have a default key, which is either key 0 or the

- OS/390 TCB key when running in emulation mode or
- PCEKEY when running in native mode.

## Implementation Approach

A VSE request (SVC x'3D' and SVC x'3E') is described by the options in register 15, the length in register 0 and the address of the subpool name in register 1. The registers of an OS/390 request are not set the same way and differ depending on the SVC/PC. So the OS/390 requests have to be mapped to the VSE interface.

- There is one entry point in the supervisor for each of the OS/390 storage management SVCs and PC.
- These entry points map the OS/390 request to the GETVIS/FREEVIS interface (R0,R1,R15) assisted by new functions incapsulated in the new module IJBSSM2. An artificial save area within the Getvis CI is used.
- BALR to the GETVIS/FREEVIS main paths  
This is done multiple times for subpool release or for variable or list request types.
- map the GETVIS/FREEVIS return codes back to the OS/390 interface via IJBSSM2

## Subpool Description

In VSE each subpool is described by an entry in the subpool index table (dsect subpint). This table cannot be extended to hold all OS/390 subpools, since it is not known in advance how many subpools are needed and the maximum number is too high.

Therefore each subpool is described by a so-called SPLE. The subpint information is part of the SPLE. The SPLE are chained together. The SPLEs are anchored to the

- TCB for task and maintask owned subpools  
TCBSPL = addr(first SPLE of task)
- SCB for address space owned subpools  
SCBSPL = addr(first SPLE of address space)
- SMCOM for system owned subpools  
SMCSYSPL = addr(first SPLE of system)

The subpool index table is extended by two entries, one to hold the current OS/390 request and one entry used for initialization.

When the first GETMAIN request is processed for a subpool during the scope of its lifetime, the subpool control block (SPLE) is created and hooked into the respective chain. This control block remains in the chain until the end of the subpool's lifetime.

As the last entry of the Subpool Index Table, at entry number MXSUBPLH+2, an additional slot is added to hold the OS/390 subpool being processed by the current request. The control block for VSE subpools is extended as shown below, to account for the characteristics of an OS/390 subpool.

At the beginning of request processing this slot is filled with the respective subpool characteristics of the MVS subpool(ACTIVATE).

At the end of request processing this slot is copied back to the respective control block describing the OS/390 subpool(UPDATE), and the slot is re-initialized.

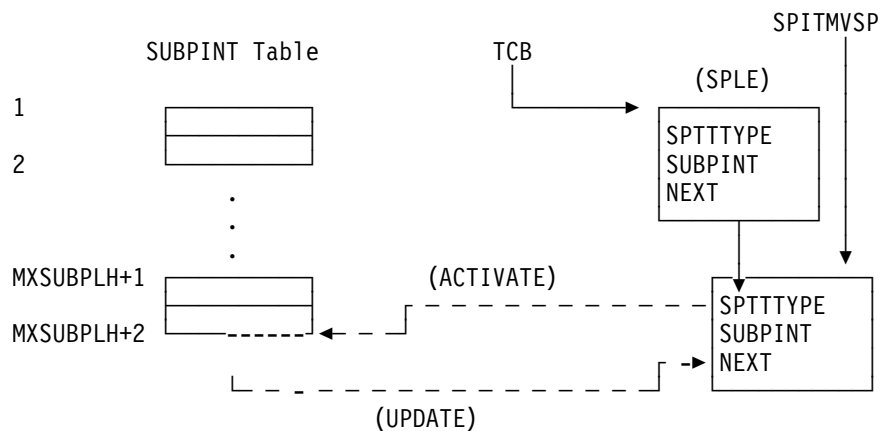


Figure 128. Control Blocks for OS/390 subpools.

## OS/390 Request Processing

- Determine area (sva,space,partition) and pfix
- Close gate(s)
- Build and anchor SPLE
- Build a 'Getvis' request (with R0,R1,RF) in artificial save area
  - let RD point to artificial save area
  - the save area is locatad within the Getvis CI
- copy SPLE in subpool index table slot
- call GETVIS/FREEVIS to process request
- update SPLE
- map VSE output to OS/390 output and pass result to user

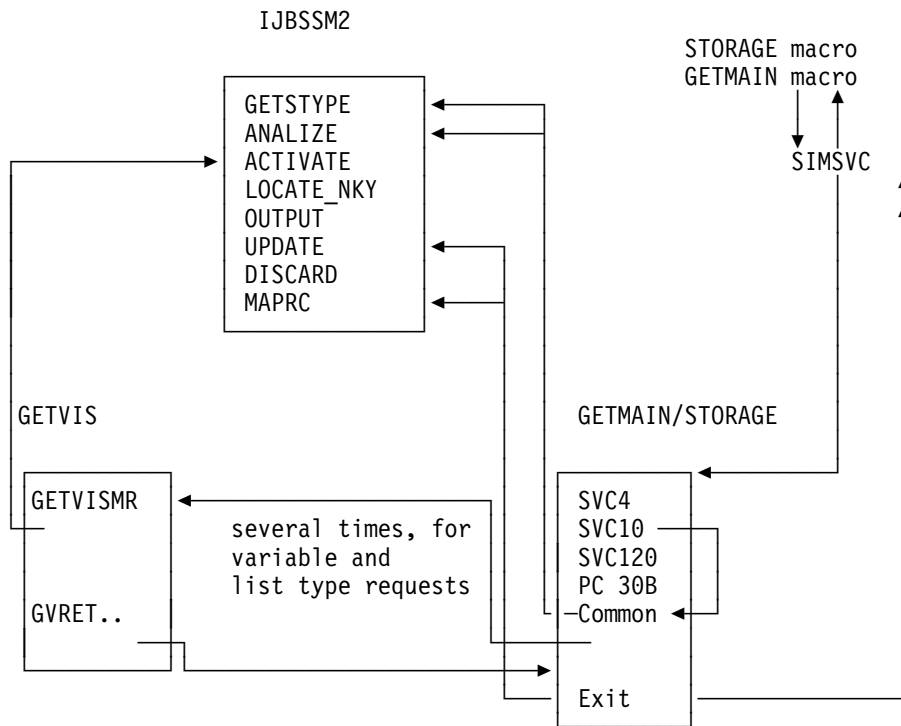


Figure 129. Code Structure for GETMAIN/STORAGE OBTAIN

1. GETSTYPE

GETSTYPE determines the area (system, space..) and whether pfix or not.

2. Gating

With the result of GETSTYPE, gating is done. The gating routine returns the ptr to the CI. The CI contains an additional save area, which is pointed to by register RD. This area must look like the VSE save area which is used during getvis processing (Getvis uses the area pointed to by RD).

- GETMAIN and PFI

During PFI processing the GETVIS gate is opened. But only one

GETMAIN request at a time may be executed, even if it is no PFI request.

The reason is that too much information would have to be saved.

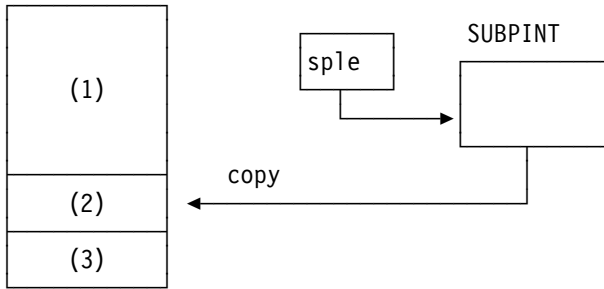
3. ANALYZE

At the end of the CI there will be as much as possible SPLEs. During ANALYZE, it is checked whether there is space for one more SPLE. If not, a return code is passed and the getmain mainpath must do a space getvis request for the SPLE.

4. With the output of the ANALYZE, a 'Getvis' request is built in the save area pointed to by register D, (R0, RF, R1(ptr to subpool name in save area)).

5. ACTIVATE

Subpool index table



- a. (1) part of subpool index table containing VSE subpools
- b. (2) slot for current MVS request
- c. (3) used to init subpool index table entry

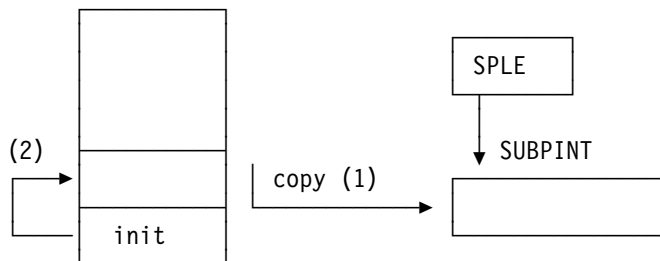
SUBPINT, pointed to by SPLE is copied. If it does not yet exist, the slot in the subpool index table is initialized.

6. OUTPUT

- 1. pass result to user
- 2. determine if list request or not

7. UPDATE

subpool index table



After the GETMAIN request is finished, the GETMAIN subpool index table entry is copied back in SUBPINT pointed to by SPLE and the element in the subpool index table is initialized by copying the init element in the subpool index table.

**Note:** When the first request of a list type request fails, all storage obtained up to now is freed again.



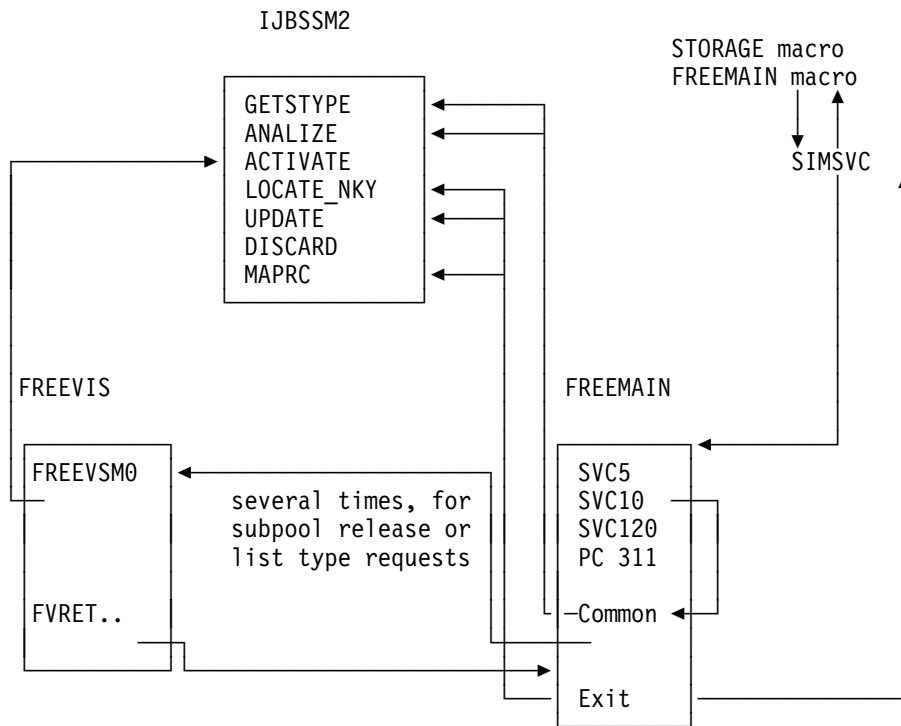


Figure 130. Code Structure for FREEMAIN.

It is assumed that a Getvis Area is initialized via a GETVIS request before the first GETMAIN/FREEMAIN request is issued (reason: GETMAIN needs initialized CI, including save area).

System Getvis : OK, done by IPL

Space Getvis : OK, done by allocation (JCL workarea)

Partition Getvis : OK, done by JCL except when partition Getvis size for EXEC REAL is zero.

If the GETVIS area does not exist, the GETMAIN request is terminated.

GETSTYPE and MAPRC do not need a save area.

**Note:** Subpool release for a subpool is the same as FREEVIS subpool except that SUBPINT is not thrown away.

FREEVSMO may be called several times for a subpool that exist several times because it has different keys.

List type request: The first request that fails terminates the list type processing. Areas freed up to now are not getmained again.

## The OS/390 Subpool Table

Figure 131 (Page 1 of 2). Storage Subpools and Their Attributes.

**Legend:**

**FP**                      *Fetch Protection*

<b>Subpool</b>	<b>Location</b>	<b>FP</b>	<b>Type</b>	<b>Owner</b>	<b>Storage Key</b>
0	Partiton Getvis	Yes	Pageable	Maintask	Same as TCB key at time of first storage request
1-127	Partiton Getvis	Yes	Pageable	Task	Same as TCB key at time of first storage request
129	Partiton Getvis	Yes	Pageable	Maintask	Selectable See note 1
130	Partiton Getvis	No	Pageable	Maintask	Selectable See note 1
131	Partiton Getvis	Yes	Pageable	Maintask	Selectable See note 1
132	Partiton Getvis	No	Pageable	Maintask	Selectable See note 1
226	System Getvis 24-bit	No	Fixed	System	0
227	System Getvis	Yes	Fixed	System	Selectable See note 1
228	System Getvis	No	Fixed	System	Selectable See note 1
229	Partition Getvis	Yes	Pageable	Task	Selectable See note 1
229	Partition Getvis	No	Pageable	Task	Selectable See note 1
230	Partition Getvis	No	Pageable	Task	Selectable See note 1
231	System Getvis	Yes	Pageable	System	Selectable See note 1
239	System Getvis	Yes	Fixed	System	0
240	Partition Getvis	Yes	Pageable	Task	Same as TCB key at time of first storage request
241	System Getvis	No	Pageable	System	Selectable See note 1
245	System Getvis	No	Fixed	System	0
250	Partition Getvis	Yes	Pageable	Task	Same as TCB key at time of first storage request

Figure 131 (Page 2 of 2). Storage Subpools and Their Attributes.

**Legend:**

**FP**                      *Fetch Protection*

Subpool	Location	FP	Type	Owner	Storage Key
251	Partition Getvis	Yes	Pageable	Maintask	Same as TCB key at time of first storage request
252	Partition Getvis	No	Pageable	Maintask	0
253	Dynamic Space Getvis	No	Fixed	Task	0
254	Dynamic Space Getvis	No	Fixed	Maintask	0
255	Dynamic Space Getvis	No	Fixed	Address space	0

**Notes:**

1. Possible storage keys are described in "Storage Keys for Selectable Key Subpools"

**Storage Keys for Selectable Key Subpools**

Figure 132 (Page 1 of 2). Possible Storage Keys

Subpool	Macros and Parameters	Storage Key
129-132	<ul style="list-style-type: none"> <li>• GETMAIN with LC,LU,VC,VU,EC or R.</li> <li>• FREEMAIN with LC,LU,L,VC,VU,V,EC,EU or R.</li> <li>• STORAGE with OBTAIN or RELEASE; CALLRKY=YES is specified.</li> </ul>	The storage key is equals the caller's PSW key. (The KEY parameter is not allowed.)
	<ul style="list-style-type: none"> <li>• GETMAIN with RC,RU,VRC,VRU.</li> <li>• FREEMAIN with RC,RU.</li> </ul>	The storage key is the key the caller specifies on the KEY parameter. If KEY is not specified, the default equals the caller's PSW key.
	<ul style="list-style-type: none"> <li>• STORAGE with OBTAIN or RELEASE; CALLRKY=YES is omitted or CALLRKY=NO is specified.</li> </ul>	The storage key is the key the caller specifies on the KEY parameter. If KEY is not specified, the default is 0.
227-231, 241	<ul style="list-style-type: none"> <li>• All GETMAIN/FREEMAIN requests</li> <li>• STORAGE with OBTAIN or RELEASE; CALLRKY=YES specified</li> </ul>	The storage key equals the caller's PSW key. (For RC, RU, VRC and VRU, the KEY parameter is ignored. For other GETMAIN and FREEMAIN requests, the KEY parameter is not allowed.)

Figure 132 (Page 2 of 2). Possible Storage Keys

Subpool	Macros and Parameters	Storage Key
	<ul style="list-style-type: none"><li>• STORAGE with OBTAIN or RELEASE; CALLRKY=YES omitted or CALLRKY=NO specified.</li></ul>	The storage key is the key the caller specifies on the KEY parameter. If KEY is not specified, the default is 0.

---

## z/Architecture Cross Memory Communication

z/VSE supports the stacking space switching Program Call.

For a detailed description of the PC-ss instruction refer to *z/Architecture Principles of Operation*, SA22-7832.

The PC-ss instruction enables programs to use cross memory communication.

---

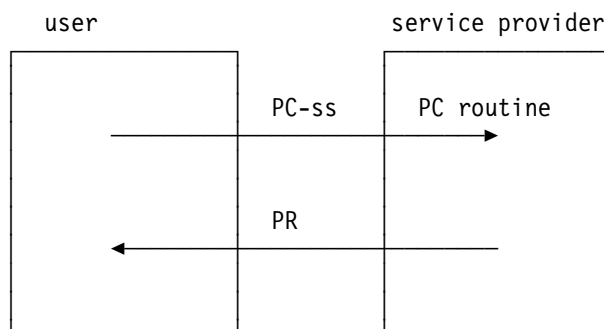
### Description

Cross memory communication enables one program to transfer control to another program. This is done by means of the PC instruction. The program that receives control is called **PC routine**. The PC routine returns to the user with the PR instruction.

The PC routine is provided by the so-called **service provider**. Before the PC instruction can be used, the necessary control block structure (environment) has to be established. This is also done by the service provider. The control block structure connects the service provider's address space to the user's address space. Part of the PC instruction is the PC number. This number determines which PC routine is selected.

The user program and the PC routine can execute in the same or in different address spaces. To transfer control to a different address space, the space-switching PC (PC-ss) must be used. However, the PC routine executes always under the same task as the user program.

A PC routine can access data in the user's address space by using access registers or by an ALET of two. With the proper authority it can also access data in other address or data spaces.



---

### The Cross Memory Environment

The cross memory environment consists of

- entry tables
- linkage tables
- PC number
- authorization tables
- program authorization - PKM (PSW Key Mask)

The services to establish the cross memory environment have been ported from OS/390. The services itself are PC based. They are available both in VSE native and OS390 emulation mode. To use the services default allocation is required, i.e

only one partition per space is allowed. The user must either execute in supervisor state or with a PKM 0.

## Entry Table (ETDEF, ETCRE service)

- contains one entry for each PC routine
- ETDEF macro to define
  - PC routine (name or address)
  - problem/supervisor state
  - space switch
  - ARR (name or address of recovery routine)
  - EAX (EAX authority for PC routine)
  - PKM
  - ...
- ETCRE macro to create entry table
  - owned by service provider's partition
  - must be connected to linkage table of user's partition

## Linkage Index LX (LXRES)

- Index in linkage table (unique within system)
- obtained by service provider through LXRES macro
- non-system LX
  - connect an entry table to selected partitions (LT) in system
- system LX (not supported in VSE)
  - connect entry table to all partitions (LT) in system
  - i.e user can not connect ETs to system linkage table in VSE

## Linkage Table (ETCON)

- each partition has a linkage table, which is the system linkage table as long as no ETCON done
- points to one or more entry tables
- entry table connected to linkage table by ETCON macro

## PC Number

- created by service provider
- consists of LX and EX (index in entry table)

## Program Authorization (PKM)

- each program (task) runs with a PSW key mask (PKM)
- 16 bit string, each bit represents storage protection key valid for a problem state program to use
- used for SPKA instruction
- defines which PC routine can be invoked
- programs are initially dispatched with a PKM equal to the partition's storage protection key
  - plus key 9 if SSP and OS390 emulation mode
- PC and PR instructions can change the the PKM

## Authorization Tables

- each partition owns an authority table
- each table entry consists of two bits
  - P-bit (PT authority)
  - S-bit (SSAR authority,EAX authority)
- each table entry corresponds to an AX (auth. index)
- AX = 0 : neither PT nor SSAR authority
- AX = 1: both PT and SSAR authority
- a partition is dispatched with an AX value of 0
- in VSE, the AX value can not be changed
- ETDEF with SASN=NEW only, i.e. neither PT nor SSAR authority needed

---

## Cross Memory Services

### AXRES - Reserve authorization index

The AX is owned by the issuing partition. The AX is used as an EAX for PC routines.

### AXFRE - Free authorization index

The AXFRE service returns an AX value to the system.

### AXEXT - Extract authorization index

The AXEXT service returns the AX of the partition. AX is always 0 in VSE.

### ATSET - Set authorization table

The service sets PT and SSAR(EAX) authority bits in the authority table for use during EAX authority checking.

### ETDEF - Create an entry table descriptor (ETD)

- ETD used as input for ETCRE to create an entry table
- defines PC routines
  - name, space switch, problem/supervisor state,EAX
  - ARR(recovery routine if PC routine abends),...

### ETCRE - Create Entry Table

- output : entry table token

### ETCON - Connect Entry Table

- connects entry table(s) to specified linkage index(es) in the linkage table of the current partition
  - pair (LX,entry table token) is used
- must run under task of service user
- connection exists until
  - ETDIS removes the connection
  - the entry table owner terminates
  - the partition that owns the linkage table terminates

## ETDIS - Disconnect Entry Table

- disconnects entry tables from the partition's linkage table
- input: entry table token(s)

## ETDES - Destroy entry table

- Only the partition that owns the ET can destroy it.
- ET must not be connected to any linkage table unless PURGE=YES is coded
- PURGE=YES disconnects the ET from all LTs of the system.
- input : ET token

## LXRES - Reserve a Linkage Index

- reserves linkage index(es).
- LX(s) owned by current partition
- LX remains reserved until
  - LXFRE frees a reserved linkage index
  - the maintask terminates

## LXFRE - Free a Linkage Index

- LX(s) must be owned by current partition
- no ET must be connected to the LX unless FORCE=YES is coded.
- FORCE=YES disconnects ET

## Cross Memory Resources

Cross Memory Resources are owned by the maintask, even if the service was given by a subtask. The resources are freed explicitly by a service or by the system at maintask termination. This is a deviation from OS/390.

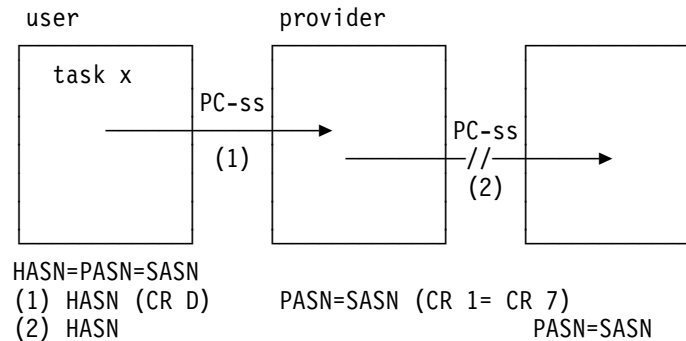
---

## Cross Memory Terminology

- ASN: address space number
  - each partition has its own ASN. In VSE the ASN is PIK/16
- Home Address Space (partition)
  - allocation space (partition), in which the task is initially dispatched.
  - CR D points to segment table of home address space
  - HASN = ASN of home space
  - during execution of a task, the home address space does not change
  - low core is set up for the home space (scbptr, pcbptr) independent of the space in which a task is currently executing
  - ALET 2 denotes the home space
- Primary Address Space
  - address space whose segment table is used to fetch instructions
  - CR 1 points to segment table of primary address space
  - PASN = ASN of primary space
- Secondary Address Space
  - used in secondary ASC mode. Not supported in VSE
  - CR 7 points to segment table of secondary address space
  - SASN = ASN of secondary space
- Cross Memory Mode
  - a task executes in cross memory mode, when the primary and home address space are different address spaces, i.e. HASN  $\neq$  PASN
- Switch to home space means



- set HASN = PASN = SASN
- it does not mean home space mode in PSW
- Space switch
  - primary address space changes
  - VSE supports SASN=NEW only, i.e. SASN changes during PC-ss
  - nested PC-ss is not supported in VSE




---

## Termination Processing - Service User

When the user partition (maintask) terminates, either normally or abnormally and all ESTAE-TYPE recovery exits and the early STXIT AB exits have been processed and did not recover the following happens:

- all subtasks are canceled due to maintask termination. When the subtask is executing in another partition by PC-ss the recovery exits (including ARR) get control. The exits must percolate (retry is not allowed).
- all entry tables connected to the LT are disconnected.
- the linkage table is freed.

---

## Termination Processing - Service Provider

The service provider partition (maintask) terminates (normally or abnormally). All AB exits have been processed and all subtasks have been terminated.

- prevent further connects to the terminating partition
- prevent further PC-ss to the terminating partition by disconnecting the provider's ETs from all LT of the system
- cancel all tasks executing in the terminating partition by PC-ss
  - an ARR will not get control
- free resources owned by terminating partition, such as ET(s), LXs, AXs.
- invalidate partition, since it is ensured that no task will access failing partition.

---

## Control Register Save Area

- control registers are partly task related and need to be saved when a task is interrupted.
- each task has two control register save areas, TCBX1CRS, TCBX2CRS in TCBXADR
- when a task is interrupted/dispatched, CRs are saved/restored depending on the RID
  - rid=x'08' or x'10': CR first save area is used

- rid=x'04' or gated: CR second save area is used
- when a task is dispatched for the first time, CRs are loaded from the CR first save area

---

## Control Register Save Area Initialization

The CR save area needs to be initialized/updated several times.

- During supervisor generation for
  - static partition maintasks and system tasks for use during IPL
- During IPL processing for
  - BG maintask and system tasks after segment table for space 0 is allocated
- During SVC 43 processing for
  - BG maintask and system tasks after ASN second table is allocated and to enable ASN translation
- During IJBLSTK processing
  - called during attach subtask processing
  - SVC 133 processing
  - TREADY COND=START processing
  - called for user tasks assigned to another partition
  - called for vendor system tasks
- Activation of system tasks
  - SGSETUP services

---

## Task Interrupt Handling

- SVC Interrupt
  - is only allowed when HASN=PASN=SASN  
exception: SVC x'79' (leave ESTAE-type exit routine)
  - otherwise the task is cancelled. During cancellation the system
    - switches to home (set PASN=HASN=SASN)
    - saves the CRs
    - dumps area around SVC
- I/O - External - Machine Check Interrupt
  - may occur in cross memory mode (PASN/=HASN)
  - is processed if non parallel state is available
  - I/O data must be in the home address space
- Page Fault Interrupt
  - when a page fault occurs, the SCB of the space where the page fault occurred is needed. Since the supervisor does not get control during a PC-ss, the SCBPTR and TIBSCB are not pointing to the active space SCB if a page fault occurs in a routine called via PC-ss (SCBPTR not changed during PC-ss). Therefore, the PMR can't use SCBPTR,
  - SCBPTR in low core can't be taken. It is not changed during PC-ss
  - SCB obtained by means of ASN translation which
    - denotes entry in ASN 2nd table (ASTE)
    - ASTE contains SCB pointer

- If the page-fault is due to usage of the 1st STD, ASN in CR4 is used.
- If the page-fault is due to usage of the 2nd STD, ASN in CR3 is used.
- The ASN 1st table origin is taken from CR14.
- The control registers used for ASN translation are taken from 1st level control register save area in TCB-extension.
- The above change requires that the following has to be true:
  - ASTE exists even for system-space (ASN=0).
  - ASN 1st table and ASTE for BG has to be allocated during ALLOCATE request for BG, to allow pagefaults in BG during INITVIRT and later on.
  - ASNs have to be given for PMR address spaces and ASTE allocated for 1st PMR address space (in \$INTVIRT), to allow ASN-translation even for PMR address spaces.
- Program Check Interrupt
  - call vendor hook (maybe in cross memory mode)
  - save status
  - set HASN=PASN=SASN
  - initiate task termination
    - if an ARR is defined, it gets control in cross memory mode



---

## z/Architecture Subsystem Storage Protection

---

### Description

When subsystem storage protection is active, key-controlled storage protection is ignored for storage locations having an associated storage-key value of 9 (see also *IBM Enterprise Systems Architecture/390 Principles of Operation*, SA22-7201.). Subsystem storage protection is set active, when the storage protection facility is installed and bit 7 of control register 0 is set. The subsystem storage protection facility can be used to protect subsystems from erroneous applications running in the same partition.

The technique for doing this is as follows. The storage accessed by the application program is given storage key 9. The storage accessed by the subsystem only is given some other non-zero key, the partition key. The application is executed with PSW key 9. The subsystem is executed with a PSW key equal to the partition key. As a result, the subsystem can access both the key-9 and the partition key storage, while the application program can access only the key-9 storage.

#### Subsystem storage protection

When the subsystem storage protection facility is installed, it is activated by VSE by setting the corresponding bit (bit 7) in control register 0.

Furthermore bit CVTOVER in the CVT is set: This bit can be used by programs to check whether subsystem storage protection is active.

Key 9 storage can be obtained by means of the GETMAIN/STORAGE macro. To do so, the program must have associated with it a PSW key mask (PKM), that indicates, that the program is authorized to use key 9.

Only programs running in **OS390 emulation mode**, are dispatched with a PKM that allows to request key 9 storage. (Of course, the PKM is only set when subsystem storage protection is active).

So programs, running in native VSE mode, cannot use the subsystem storage protection facility.

EXEC ...,OS390 is rejected if there is more than one partition per space.

So only one partition within one address space can have key 9 storage.

Within a partition, it is now possible to have different keys:

- partition key storage
- key 9 storage
- key 0 storage

**Note:** Partition F4 has partition key 9. Therefore subsystem storage protection cannot be used in partition F4.



---

# z/Architecture Access Registers

---

## Introduction

Access register support will be used

- for cross memory services by authorized programs
- to increase the addressing capability of programs

**Note:** Access registers were introduced with ESA/390. Only minor changes in z/VSE's access register support were necessary to support z/Architecture mode. These changes are:

- CR0.47 (CR0.15 in ESA/390) is no longer used. The only format supported is the one, that was set in ESA/390 with CR0.15 = 1. When running in ESA mode, VSE always had set CR0.15=1, thus using the format (e.g. 64-bytes ASTE), which is now the only one supported. CR0.47 is no longer set by z/VSE.
- The layout of DU-AL and ASTE has changed in z/Architecture.

## Access Register Translation (ART)

Reference: *z/Architecture Principles of Operation*, SA22-7832.

z/Architecture provides a set of registers known as the Access Registers. There are 16 32-bit Access Registers (numbered 0-15) and their usage is paired with the General Purpose Registers.

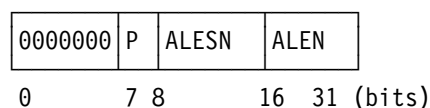
The Access Registers are used to provide Dynamic Address Translation (DAT) with a different Segment Table Origin (STO) during address translation. The contents of the Access Register do not contain the STO, but contain a Access List Entry Token (ALET). This ALET is used by the ART to verify authorization to the space and to complete the address translation. As with DAT, a look-aside buffer is provided to improve the performance of the address translation.

Once a program has obtained the proper ALET from the z/VSE supervisor, the program itself controls when it is to be in Access-Register mode and which Access Register(s) are to be used.

Only data references through a base register are affected by Access Registers. Instructions are always fetched from the primary address space. Access Registers do not apply to index registers. Since General Purpose Register zero can never be used as a base register, Access Register zero is never used for ART.

Access Register Translation (ART) uses the following control blocks and fields:

**Access-List Entry Token (ALET):** The ALET has the following format:



A DSECT (MAPALET) is provided to map the ALET.

*Primary-List Bit (P)* specifies which Access List contains the designated Access List Entry:

- 0 - Dispatchable-Unit Access List
- 1 - Primary-Space Access List

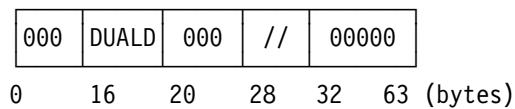
The specified list is called the Effective Access List.

*Access-List-Entry Sequence Number (ALESN)* is described on page 310.

*Access-List-Entry Number (ALEN)* when multiplied by 16 is the number of bytes from the beginning of the effective access list to the designated Access-List Entry.

The ALET is placed into the appropriate Access Register by the program prior to placing itself into Access-Register Mode.

***Dispatchable-Unit Control Table (DUCT)***: The DUCT is pointed to by Control Register 2 and must be on a 64-byte boundary. The format of the DUCT is:

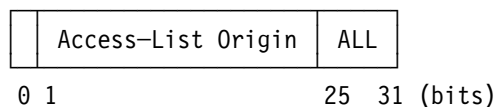


Bytes 0-15, 20-27 and 32-63 of the DUCT are reserved for possible future expansions and should contain all zeros. Bytes 28-31 are available for use by programming.

A DSECT (MAPDUCT) is provided to map the DUCT.

*Dispatchable Unit Access-LIST Designation (DUALD)* contains the Access-List Designation (ALD). The dispatchable-unit and primary-space access-list designations both have the same format:

Format-0 Access-List Designation

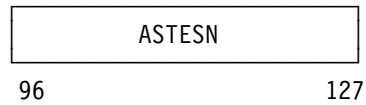
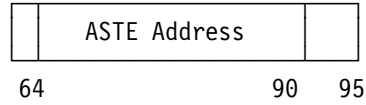
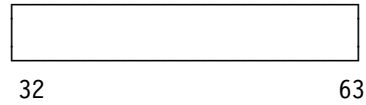
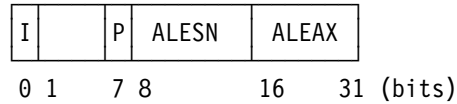


Bit 0 is reserved for a possible future expansion and should be zero.

The Access-List must be aligned on a 128-byte boundary. The Access-List Length (ALL) specifies the length of the Access-List in units of 128 bytes (making the length of the Access-List variable in multiples of eight 16-byte entries. The length of the Access-List, in unit of 128 bytes, is one more than the value in bit positions 25-31.

***Access-List Entries***: The effective Access List is the Dispatchable-Unit Access List if bit 7 of the ALET being translated is zero, or is the Primary-Space Access List if bit 7 is one. The entry fetched from the effective list is 16 bytes in length and has the following format:





A DSECT (MAPALE) is provided to map the ALE.

Bits 1-6, 32-64 and 90-95 are reserved for possible future expansions and should be zeros.

In both the Dispatchable-Unit Access List and the Primary-Space Access List, Access-List entries 0 and 1 are intended not to be used in Access-Register Translation (ART). Bits 1-127 of Access-List Entry 0 and bits 1-63 of Access-List Entry 1 are reserved for possible future expansion and should be zeros. Bit 0 of Access-List Entries 0 and 1, and bits 64-127 of Access-List Entry 1, are available for use by programming. However, bit 0 of Access-List Entries 0 and 1 should be set to one in order to prevent the use of these entries in which the ALEN is 0 or 1.

The fields in the Access-List Entry are:

*ALEN-Invalid Bit (I)*: Bit 0, when zero, indicates that the Access-List Entry specifies an address/data space.

*Private Bit (P)*: Bit 7, when zero, specifies that any program is authorized to use the access-list entry in access-register translation. When bit 7 is one, this indicates that further authorization is required and described below.

*Access-List-Entry Sequence Number (ALESN)*: These bits are compared against the ALESN in the ALET during ART. They must be equal for ART to continue.

*Access-List-Entry Authorization Index (ALEAX)*: Used to determine whether or not a program using access-register translation is authorized to use the Access-List entry. The program is authorized if any of the following conditions are met:

1. Bit 7 is zero (indicating public)
2. ALEAX matches Extended Authorization Index (EAX) in Control Register 8.
3. The EAX selects a secondary bit that is one in the authority table for the specified address space.

*ASN-Second-Table-Entry (ASTE) Address*: Points to the ASTE for the specified address space. ART obtains the STO for the address space from the ASTE.

*ASTE Sequence Number (ASTESN)*: Compared against the ASTESN in the designated ASTE during ART. An inequality halts ART.

**ASN-Second-Table Entries (ASTE)**: The ASTE begins on a 64 byte boundary. Each entry is 64 bytes in length. The layout is described in *z/Architecture Principles of Operation*, SA22-7832 ASN-Second-Table Entries. Since z/VSE works with segment tables only (no region table), ASCE part 1 is zero. ASCE part 2 contains the address of the segment table (as in CR 1).

A DSECT (MAPASTE) is provided to map the ASTE.

*ASX-Invalid Bit (I)*: Bit 0 controls whether the address/data space associated with the ASTE is available. When bit 0 is one, ART is halted.

*Authority-Table Origin (ATO)*: Designates the beginning of the authority table. This is used only if the private bit in the access-list entry is one and the access-list-entry authorization index (ALEAX) in the access-list entry is not equal to the EAX in CR8.

*Authorization Index (AX)*: Not used during ART.

*Authority-Table Length (ATL)*: Specifies the length of the authority table in units of four bytes. If the EAX is greater than the ATL, then ART is halted if extended authority must be checked.

*Segment-Table Designation (STD)*: Used by DAT to translate the logical address for the storage-operand reference being made during ART.

*Linkage-Table Designation (LTD)*: Not used during ART.

*Access-List Designation (ALD)*: When this ASTE is designated as the Primary ASTE Origin in Control Register 5, this field becomes the Primary Access-List designation (PSALD).

*ASN-Second-Table Entry Sequence Number (ASTESN)*: Compared against the ASTESN in the Access-List Entry. An inequality halts ART.

Bits 224-255 (bytes 28-31) are available for use by programming.

The second 32 bytes of the 64-byte ASTE are reserved for possible future extensions and should contain all zeros.

**Access-Register-Translation (ART) Process:** ART operates on the access register designated in a storage-operand reference in order to obtain a segment-table designation for use by DAT. When one of the access registers 1-15 is designated, the Access-List-Entry Token (ALET) that is in the Access Register is used to obtain the Segment-Table designation. When access register 0 is designated, an ALET having the value of 00000000 hex is used.

When the ALET is 00000000 or 00000001 hex, the Primary or Secondary Segment-Table designation, respectively, is obtained.

When the ALET is other than 00000000 or 00000001 hex, the Primary-List bit in the ALET is used and the contents of Control Register 2 or 5 are used to obtain the effective Access-List designation and the Access-List Entry Number (ALEN) in the ALET is used to select an entry in the effective Access List.

The Access-List Entry is checked for validity and for containing the correct Access-List-Entry Sequence Number (ALESN).

The ASN-Second-Table Entry (ASTE) addressed by the access-list entry is checked for validity and for containing the correct ASN-Second-Table-Entry sequence number (ASTESN).

Whether the program is authorized to use the Access-List Entry is determined through the use of one or more of:

1. The private bit and Access-List-Entry Authorization Index (ALEAX) in the Access-List Entry
2. The Extended Authorization Index (EAX) in Control Register 8, and
3. An entry in the Authority Table addressed by the ASN-Second-Table Entry.

When no exceptions are recognized, the Segment-Table designation in the ASN-Second-Table Entry is obtained.

To improve the performance of ART, an ART-Lookaside Buffer (ALB) is provided.

**Access Register Indication on Page Fault:** When a segment- or page-translations exception occurs on an System z processor, the processor must indicate to the software how the virtual address was translated (using the primary STD, secondary STD, home STD, or in access-register mode). This is indicated in the last two bits of the 8-bytes *Translation-Exception Identification* field at location 168 (x'A8') (in z/VSE, bytes 168-171 are zeros, since z/VSE supports virtual addresses up to 2GB only).

- 00 - Primary STD used
- 01 - Access register mode
- 10 - Secondary STD used
- 11 - Home STD used

**Note:** z/VSE maintains the ESA/390 location x'90' - x'93'.

When access-register mode is indicated and the translation exception was caused by a storage-operand reference that used an AR-specified STD, then the specific access register being used is indicated in low storage field called *Exception Access Identification* at location 160 (x'A0'). Location 160 (x'A0') contains zeros if the processor was in access-register mode and the translation exception was caused by an instruction fetch.

0000	xxxx
------	------

 – Access Register Number  
x'A0'

*Translation Modes:* z/Architecture offers four modes of operation:

- Primary-Space Mode
- Secondary-Space Mode
- Access-Register (AR) Mode
- Home-Space Mode

Home-Space Mode can only be set in supervisor state. The other translation modes can be set by the user. The design in z/VSE is such that effectively only Primary-Space Mode and Access-Register Mode are used. To avoid use of Secondary-Space Mode, z/VSE will always insure that Control Registers 1 and 7 are always equal. A program using the Set Address Control (SAC) instruction to set itself in Secondary-Space Mode will not fail, but will always be addressing data in its own address space.

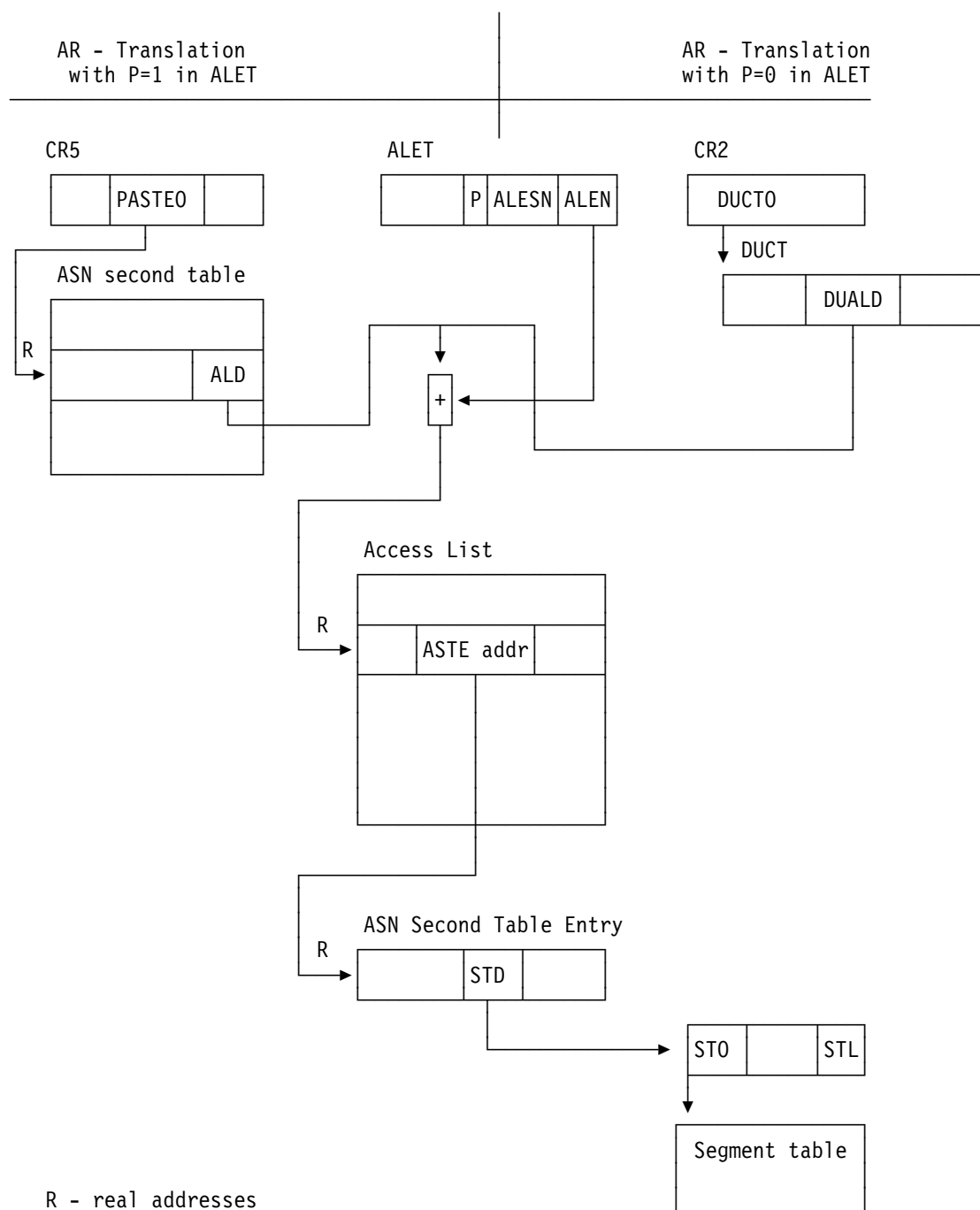


Figure 133. Access Register Translation

## z/VSE Implementation

Two types of Access Lists are supported in z/Architecture

- A primary address space access list (PASN-AL) - addressed through CR 5.
- A dispatchable unit access list (DU-AL) - addressed through CR 2

z/VSE has a dummy DUCT, pointing to a dummy DU-AL, where all entries are invalid.

In z/VSE, each dispatchable work unit (that means, a VSE task) has a DU-AL. When the task is initialized, it gets the dummy DU-AL. When the task adds the first entry to the DU-AL, a DU-AL for the task is created. This DU-AL is deleted when the task is terminated. If the task is created with the ATTACH macro using the ALCOPY=YES parameter the DU-AL is a copy of the DU-AL of the attaching task, in all other cases an empty DU-AL is created. The DU-AL can contain entries for data spaces TYPE=SINGLE|ALL and address spaces.

When the system allocates a partition, it gives that partition a PASN-AL that contains entries for all currently defined common data spaces (that means, data spaces created with parameter SCOPE=COMMON).

The PASN-AL is divided into two parts. The first part is designed to contain entries for data spaces defined with TYPE=COMMON. The number of entries in this 'common' part of the PASN-AL is the sum of

- three reserved entries for hardware purpose
- five entries default value for data spaces TYPE=COMMON. This value can be modified with the SYSDEF AR command or the SYDEF JCL statement.
- number of VDISKS added at IPL time.

The second part (private part) of the PASN-AL is designed to contain entries for data spaces TYPE=SINGLE|ALL and address spaces.

If a program needs access to data in another address space or data in a data space, a connection between the program and the address/data space must be established that means, an entry in either the DU-AL or the PASN-AL must be created and the program has to know the ALET that indexes the entry in the AL.

The entry in the AL and the related ALET may be obtained either through a GETFLD FIELD=ALET macro call in case of an address space or through an ALESERV macro call in case of address space and data space.

When the first request after IPL for an ALET is issued in the system (either ALESERV or GETFLD) a 'Model PASN-AL' is created. This Model will be given to partitions at allocation time. As long as the partition does not add entries to the private part of the PASN-AL it will stay with the Model PASN-AL.

When a partition is deallocated the related PASN-AL is invalidated (if it is not the Model PASN-AL) and all entries pointing to the partition in all ALs are invalidated.

The Access Registers will be saved/restored at entry/exit to the supervisor. The proper EAX value will be loaded into CR 8 when the task is dispatched.

The page fault address will be handled properly if Access Register Mode was in effect when the page fault occurred.

The partition's ASTE is updated to point to the proper address space's SCB when the partition switches modes (REAL|VIRTUAL).

The partition's Access List Entries are invalidated when the partition is unbatched. For dynamic partitions, this occurs at end of VSE/POWER job. For static partitions, this occurs only when the partition is explicitly unbatched.

*SDAIDS* will recognize that Access Register Mode was in effect when the data reference was made to a data area that PER was monitoring.

*DUMP* will dump and print the 16 Access Registers along with the General Purpose Registers.

*Authorization:* Public Access List Entries are used for data spaces that means, no authorization check is performed for data spaces by ART. Access List Entries for address spaces are made private in z/VSE, forcing ART to perform validation before granting access to the ALE. VSE supports two methods to set the value for the EAX (CR 8) so that ART will be successful.

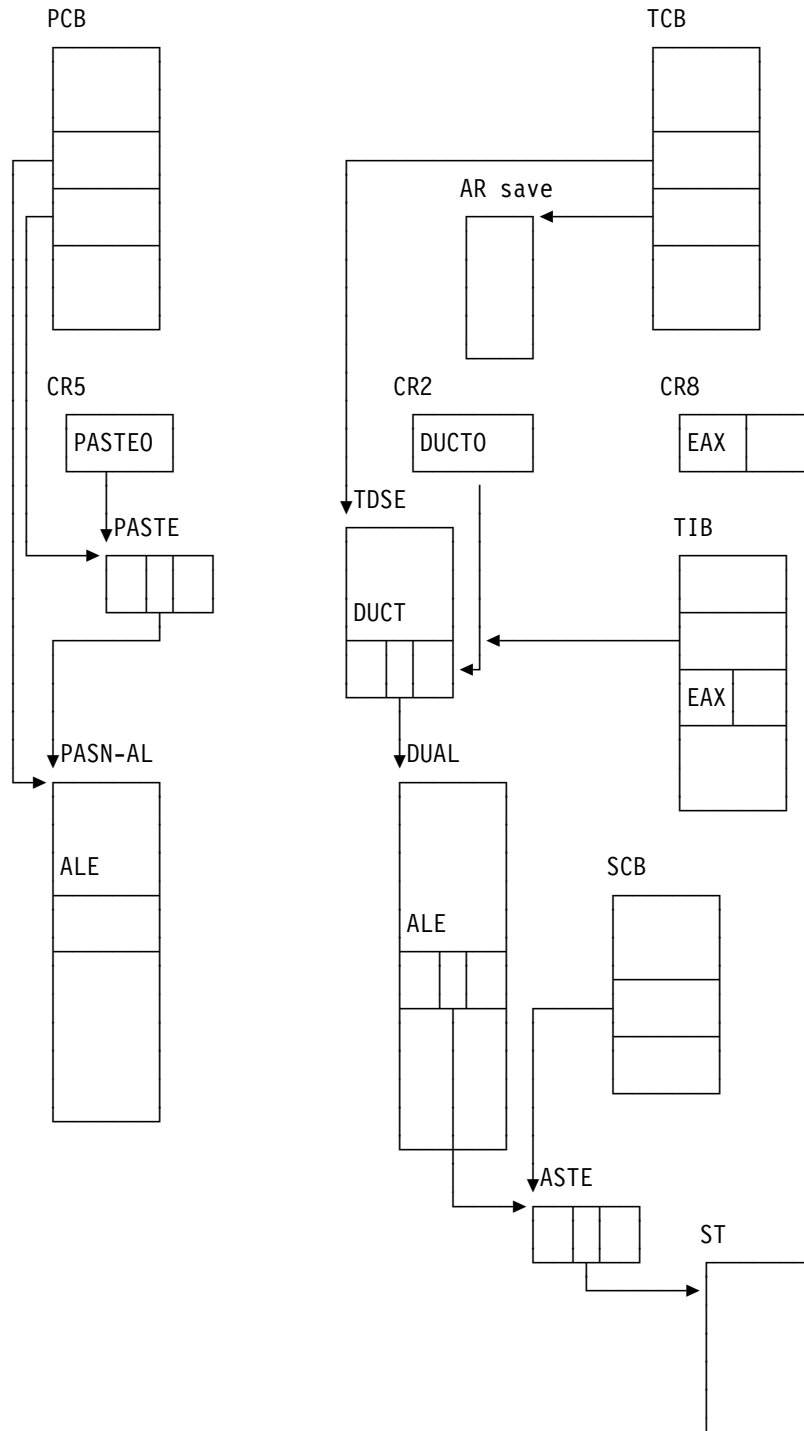
- GETFLD FIELD=ALET to load a system defined EAX into CR8 of the current task.
- ETDEF and ETCRE to set the EAX in an entry in the entry table for a specific space switching PC (program call instruction). Execution of the PC will load the EAX from the entry table entry into CR8.

Extended authorization check is setup to fail always: Authority Table Designation points to a field of binary zeros.

**ART Control Block Structure in z/VSE:** Each partition has its own ASTE. The PCB contains a pointer to the ASTE. The ASTEs for the static partitions are allocated during supervisor generation. The ASTE for a dynamic partition is allocated at the time the dynamic partition is allocated.

The TCB contains a pointer to a task's access registers save area. The access register save areas for the static partitions (maintasks) are allocated during supervisor generation. The access register save area for a subtask or dynamic partition is allocated when the subtask or dynamic partition is allocated.

The control block structure implemented in z/VSE is:





---

## Address Spaces

The usage of Access Registers in z/VSE is for cross memory services for authorized programs to provide efficient means of accessing data that resides in other address spaces. Authorized programs have to introduce themselves to the VSE system either by means of a SUBSID macro call or a PRODID macro call.

Authorization by means of the PRODID macro is designed to be used by Vendor programs.

The authorized programs using SUBSID are:

- VSE/POWER
- ACF/VTAM
- OCCF
- VSE/PT
- CICS TS<sup>3</sup>

To use Access Registers to address another address space, the authorized program must do the following:

- Obtain the
  - PIK of the target partition (GETFLD service is available) if GETFLD FIELD=ALET is used
  - STOKEN of the target partition may be retrieved by the target partition and must be passed to the calling partition if ALESERV ADD is used
- To obtain the ALET for that partition use
  - GETFLD FIELD=ALET,PART=pik
  - ALESERV ADD with STOKEN for partition
- Load the desired Access Register with the ALET.
- Set itself into Access-Register Mode by using the SAC 512 instruction.
- Reference the data using any ESA/390 instruction except MVCP or MVCS (key zero may be required).
- Take itself out of Access-Register Mode by using the SAC 0 instruction.

A z/VSE service (XMOVE) is provided which allows movement of data between partitions in the same or other address spaces. To use the XMOVE facilities, the program would do the following:

- Obtain the PIK of the partition that the authorized program wants to address (GETFLD services are provided for this purpose).
- Use the GETFLD FIELD=ALET,PART= service to obtain the ALET for that partition.
- Place the ALET into the desired ALET field for the XMOVE parameter list (XMOVE provides for both 'from' and 'to' ALET fields).
- Issue the XMOVE macro.

---

<sup>3</sup> Support for CICS TS subsystem is provided primarily so that transactions can be written to access data in other address spaces. CICS TS contains no knowledge of the access registers. It is the responsibility of the user transaction to save the access registers prior to going to CICS for services and to restore those access registers on return. It is also the responsibility of the transaction to ensure that access-register mode is set off prior to going to CICS for services.

The GETFLD FIELD=ALET,PART=pik service is no fast path SVC anymore. It is a service-class C request now, that means, a normal SVC which can be issued only by authorized programs in user state (RID=8). The SVC checks if the requestor is a authorized subsystem. If so, EAX in ASTE is set up. Then the input for calling phase IJBALE is build:

- Register 5 points to the PCB of the partition to which access should be established.
- Register 14 will be loaded with a function code 1 if the request comes from the VTAM partition, function code 2 if the request is issued by a VTAM application and function code 0 in all other cases. IJBALE will build an entry in the DUAL of the current task if the function code is 0. If the function code is 1 or 2 an entry in the private part of VTAM's PASN-AL is build. If the function code is 0 or 1 the EAX is loaded from the ASTE into CR8.
- Call IJBALE.
- Pass ALET in register 1 and return code in register 15 to issuer of GETFLD.

The EAX in CR8 is only set up properly by means of calling IJBALE with function code 0 or 1. Therefore, when the effective access list is created with ATTACH ALCOPY=YES, authorization check of ART will fail as long as no GETFLD FIELD=ALET is issued from the current task.

---

## Data Spaces

### Introduction

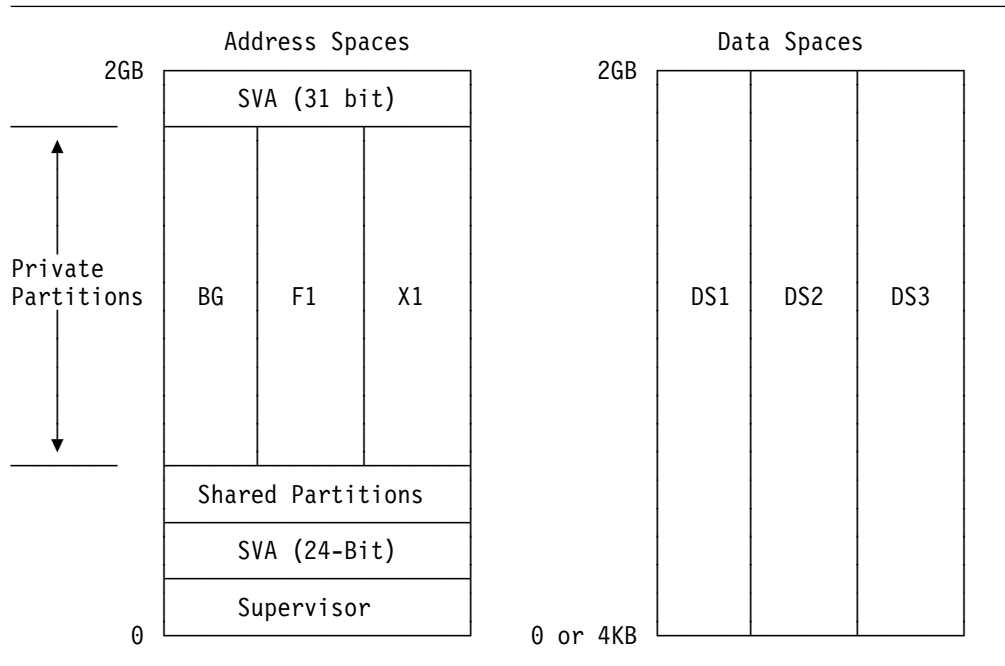
An address space, literally defined as the range of addresses available to a computer program, is like a programmer's map of the virtual storage available for code and data. An address space provides each programmer with access to all of the addresses available through the computer architecture.

Because it maps all of the available addresses, an address space includes system code and data as well as user code and data. Thus, not all of the mapped addresses are available for user code and data. This limit on user applications was a major reason for System/370 Extended Architecture (370-XA). Because the effective length of an address field expanded from 24 bits to 31 bits, the size of an address space expanded from 16 megabytes to 2GB.

A 2GB address space, however, does not, in and of itself, meet all of programmers' needs in an environment where processor speed continues to increase, where businesses depend on quick access to huge amounts of information stored on DASD.

What programmers need in this environment is a large address space, of course, but, even more, programmers need the ability to control what goes on in all those addresses. Extended addressability meets that need. It allows programmers to extend the power of applications through the use of additional address spaces or data-only spaces. The data-only spaces that are available for your programs are called data spaces. Your program can ask the system to create these spaces. Their size can be up to 2GB, as your program requests. Unlike an address space, a data space contains only user data; it does not contain system control blocks or common areas. Program code cannot run in a data space.

The following diagram shows, at an overview level, the difference between an address space and a data space.



## Invocation.

Data Space services are invoked via the macros

- DSPSERV to create, delete, extend or release a data space.
- ALESERV to control access to a data space.
- SYSDEF to modify installation limits for data spaces. Or to retrieve data space information for JCL and sub-systems. For the description see Appendix B.

## DSPSERV Macro

The DSPSERV macro expands into the definition of the following parameter list and a Program call (program call numbers X'00000900' and X'00000903'). The Program Check Handler gets controls and interprets the Program Call 208 by passing control to the SVA routine IJDSP.

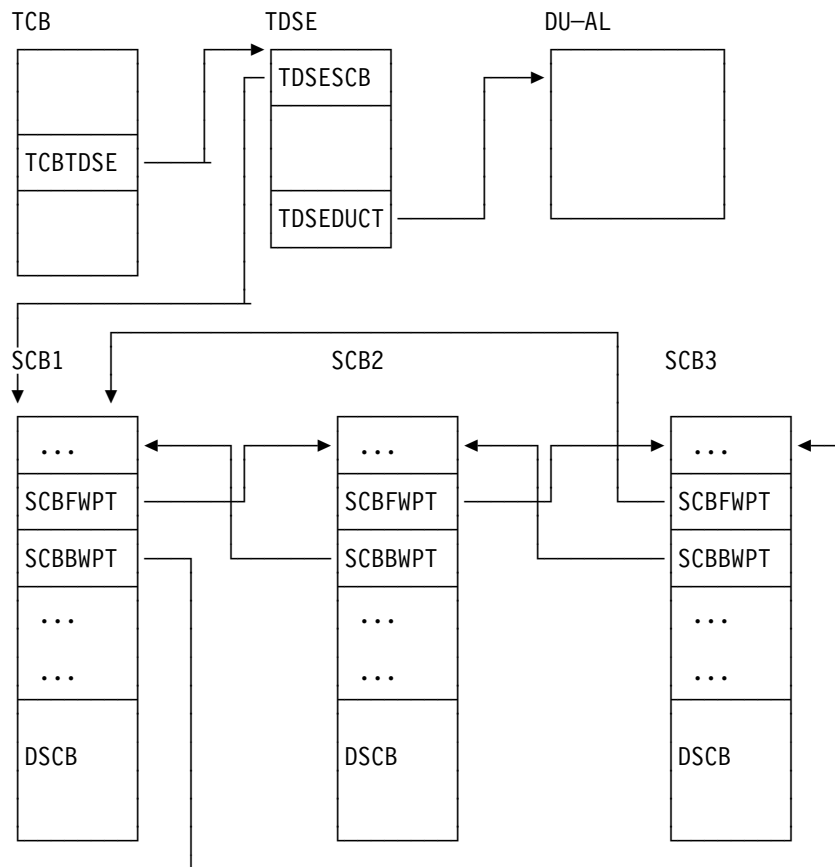
Bytes		Label	Description
Dec	Hex		
0 - 0 1 - 1	0 - 0 1 - 1	DSPXVERS DSPXSERV	VERSION IDENTIFIER SERVICE CODE: X'01' CREATE X'02' DELETE X'03' RELEASE X'06' EXTEND
2 - 2 3 - 3	2 - 2 3 - 3	DSPXFLG1	Must be zero FLAGS: B'100xxxxx' SCOPE=SINGLE B'010xxxxx' SCOPE=ALL B'001xxxxx' SCOPE=COMMON B'xxxx1xxx' GENNAME=COND B'xxxx1xx' GENNAME=YES B'xxx0xx00' MUST BE ZEROES
4 - 4	4 - 4	DSPXFLG2	FLAGS: B'01000000' FETCH PROTECT B'0x000000' MUST BE ZEROES
5 - 5 6 - 6	5 - 5 6 - 6	DSPXKEY DSPXTYPE	STORAGE KEY ONLY TYPE= BASIC SUPPORTED
7 - 7	7 - 7	DSPXFLG3	X'80' FLAGS: X'00' MUST BE ZERO
8 - 15 16 - 23 24 - 27 28 - 31 32 - 47 48 - 51 52 - 55 56 - 59	8 - 10 11 - 17 18 - 1B 1C - 1F 20 - 2F 30 - 33 34 - 37 38 - 3B	DSPXOUTN DSPXSTKN DSPXSTRT DSPXBLKS DSPXTTKN DSPXORIG DSPXNBLK DSPXINIT	NAME OF DATA SPACE STOKEN START ADDRESS FOR RELEASE NUMBER OF MAXIMAL BLOCKS TTOKEN START ADDRESS OF DSPACE NUMBER OF EXTENDED BLOCKS NUMBER OF INIT BLOCKS OF DAT

## ALESERV Macro

The ALESERV macro expands into a declaration of the following parameter list and a PC instruction (program call numbers X'0000000D' for ADD, X'0000000E' for DELETE, X'0000000F' for EXTRACT and X'00000010' for SEARCH)

Bytes		Label	Description
Dec	Hex		
0 - 0	0 - 0	ALSSRVC	SERVICE CODE: X'01' ADD X'02' ADD-PASN X'03' DELETE X'04' EXTRACT X'05' SEARCH X'06' EXTRACTH
1 - 1	1 - 1	ALSFLGS1	FLAGS: B'00000xxx' RESERVED B'xxxxx100' MUST BE 1 B'xxxxxx0x' DU-AL B'xxxxxx1x' PASN-AL B'xxxxxxx0' MUST BE 0
2 - 2	2 - 2	ALSRESV1	reserved: must be 0
3 - 3	3 - 3	ALSRESV2	reserved: must be 0
4 - 7	4 - 7	ALSALET	ALET
8 - 15	8 - F	ALSSTOKN	STOKEN

## Control Block Structure.



Every data space is owned by the task which created it. In the TDSE there is a pointer (TDSESCB) to the SCB of the data space which was first created by the task. All SCBs for data spaces which were created later on are chained forward and backward amongst each other. The forward pointer of the last SCB in this chain points to the first SCB in the chain. The backward pointer of the first SCB in this chain points to the last SCB in the chain. TDSESCB is X'00000000' when the task does not own any data space.

- Extension of SCB for Data Spaces:

For every Data Space a Space Control Block is created as it is done for Address Spaces. But for Data Spaces this control block has an extension. The format of the extension is described in the following figure. Note that this extension is only appended to the SCB, if the flag SCBDSP within the SCB is on.

Bytes		Label	Description
Dec	Hex		
128 – 135	80 – 87	DSCBNAME	NAME OF DATASPACE
136 – 143	88 – 8F	DSCBSTKN	STOKEN OF DATA SPACE
144 – 146	90 – 92	DSCBFLGS	FLAGS B'100xxxxx' SCOPE = SINGLE B'010xxxxx' SCOPE = ALL B'001xxxxx' SCOPE = COMMON B'xxx1xxxx' VDISK B'xxxx0000' RESERVED
147 – 147	93 – 93	DSCBSKEY	STORAGE KEY OF DATA SPACE
148 – 151	94 – 97	DSCBSTRT	ADDRESS OF AREA TO RELEASE
152 – 155	98 – 9B	DSCBRLSZ	SIZE OF AREA TO RELEASE
156 – 156	9C – 9C	DSCBREND	END INDICATOR FOR RELEASE
157 – 159	9D – 9F	DSCBRSVD	RESERVED
160 – 163	A0 – A4	DSCBTIB	ADDRESS OF OWNER'S TIB

SCBs for data spaces are always located in the system GETVIS area (31-bit).

- Extension of TCB for Data Spaces:

All the task specific fields needed for the data space support are defined in a separate control block within the system GETVIS area (31-bit) addressed via the field TCBDSE (within TCB).

Bytes		Label	Description
Dec	Hex		
0 – 3	0 – 3	TDSEID	IDENTIFIER OF CONTROL BLOCK
4 – 7	4 – 7	TDSESCB	ADDRESS OF FIRST DATA SPACE IN CHAIN OF OWNED DATA SPACE
8 – 11	8 – B	TDSEAUTS	ADDRESS OF AUTOMATIC STORAGE FOR IJBDSP/IJBALE
12 – 15	C – F	TDSEAUTL	LENGTH OF AUTOMATIC STORAGE FOR IJBDSP/IJBALE
16 – 19	10 – 13	TDSETIB	TIB ADDRESS OF CURRENT TASK
20 – 63	14 – 3F		RESERVED
64 – 127	40 – 7F	TDSEDUCT	DISP. UNIT CONTROL TABLE





---

# z/Architecture Linkage Stack

---

## Introduction

z/Architecture offers the facility to call subroutines or programs where neither the caller nor the called program has to make provisions to save registers, access registers, address space numbers etc.

To support BAKR and stacking PC instructions, z/VSE

- provides the necessary storage area (called linkage stack)
- generates Linkage Table and Entry Tables for selected stacking PCs
- loads control register 13 with home segment-table designation
- loads control register 15 with virtual linkage-stack-entry address

Linkage stack instructions and layout of the linkage stack are described in:

*z/Architecture Principles of Operation, SA22-7832.*

**Note:** The Linkage stack was introduced with ESA/390. Only minor changes in z/VSE's linkage stack support were necessary to support z/Architecture mode. The functionality has not changed.

These changes are:

- Adaptations to the changed layout of the linkage stack entries.
- When allocating getvis storage for a linkage stack, consider the increased length of a status entry. The status entry has changed to support 16-bytes PSWs, and 8-bytes addresses and registers

---

## Linkage Stack - z/Architecture implementation

When a task is started it gets a linkage-stack, consisting of a header, one status and a trailer entry. The linkage stack for all system tasks and BG is generated within supervisor. This allows system tasks and BG to use stacking PC during IPL.

When the linkage stack becomes full program check is raised (a 'Stack-Full Exception' X'0030' or X'00B0' at location X'8E'). If the Stack-Full exception occurs while RID was 4 (REENTRID), 8 (USERTID) or 16 (RESVCID), the program check handler saves the status into the appropriate save area and calls then phase \$IJBLSTK which assigns a linkage stack in the System-Getvis-Area and reexecutes the failing BAKR or stacking PC instruction.

When a task terminates, \$IJBLSTK is also called to free up the linkage stack space in the System-Getvis-Area and give the task the dummy-linkage-stack.

z/VSE supports two types of linkage stacks. The 'normal linkage stack' which contains 97 status entries for use by the programs running under a single z/VSE task. The task is canceled when it requests more than 97 status entries. A 'recovery linkage stack' is created for the task, which is used during abend-exit processing. It consists of 24 entries. If the abend-exit routine requires more than 24 entries in the recovery linkage stack, the task is finally terminated.

---

## **\$IJBLSTK - Create/Modify/Delete linkage stack**

\$IJBLSTK is designed to handle all requests dealing with the linkage stack. The phase is pfixed and resides in the SVA high. Several function codes are defined to handle the different tasks which may be requested from \$IJBLSTK.

### **Function code 0 - create/extend linkage stack**

This function is called from the program check handler (SGPCK) when a Stack-Full exception was raised.

Its processing consists of the following steps:

- If the recovery linkage stack has already 24 entries, cancel the task (ERR47).
- If the linkage stack has already 97 entries prepare exit to cancel task, save normal linkage stack entry address, set up dummy linkage stack and indicate that recovery linkage stack is active from now on.
- Get a new linkage stack section consisting of a header entry, 12 status entries and a trailer entry out of a linkage stack pool.  
Extend pool if possible (2nd save area is free), otherwise inform supervisor to extend pool (IJBLPOOL).
- Append new section to trailer entry of current section.
- Initialize first control register save area for CR15 in TCB-extension (TCBX1CRF) and load CR15.
- Exit to dispatcher or DISPSERV (to cancel user)

### **Function code 4 - Delete Recovery Linkage Stack, Empty Linkage Stack.**

This function is called from

- Terminator (SGAP) before clean-up routines get control
- Dispatcher (DISP) at termination of system tasks
- AB exit processing (SVC 95, SGEXIT)

Its processing consists of the following steps:

- clear second linkage stack savearea (TCBX2CRF in TCB-extension)
- clear VTAMs linkage stack savearea (TCBX1TLA in TCB-extension)
- If recovery linkage stack exists issue SFREEVIS for all its linkage stack sections, update TCB-extension and reestablish normal linkage stack.
- set TCBX1CRF to point to header of first section of normal linkage stack, load cr15 and return to caller

### **Function code 8 - Initialize Linkage Stack Values**

This function is called from

- Terminator (SGAP, DISPSERV fct.code 36) after VENDOR EOT hooks
- Attach subtask (SGAP, SVC 38)
- Start program (SGAP, SVC 133)

Its processing consists of the following steps:

- clear second linkage stack savearea (TCBX2CRF in TCB-extension)
- clear VTAMs linkage stack savearea (TCBXTLSA in TCB-extension)
- If recovery linkage stack exists issue SFREEVIS for all its linkage stack sections.
- Give task normal linkage stack consisting of a header, one status and a trailer entry and issue SFREEVIS for all other sections.
- Initialize linkage stack related fields in TCB-extension from the Data Space Information Control Block.
- set TCBX1CRF to point to header of the linkage stack, load cr15 and return to caller

## **Function code 12 - Free storage for both linkage stacks**

This function is called from SGTINF for TSTOP COND=UNBATCH

It issues SFREEVIS requests for all existing linkage stack sections (except the permanent sections i.e. the normal linkage stack consisting of a header, one status and a trailer entry).



---

# Capacity Measurement Tool (CMT) in z/VSE 4.1

---

## Introduction

Capacity Measurement is started in any partition through:

```
// EXEC IJBCMT,PARM='START ID=xxxx'
```

When started, every 30 minutes it writes a so-called SCRT89 record into a sequential disk file. The data in the SCRT89 record are similar to the one in the SMF70 record of a z/OS system. CMT is implemented as a system task, i.e. collecting of measurement data and writing of the SCRT89 record into the disk file is done under control of a system task.

---

## Characteristics of the CMT system task

- The system task is named CMT
- CMT task id is x'0D'. This is a reuse of the REC system task, which was not used.
- Priority of the CMT task is between DSP and SPT task
- The CMT task is generated with TIBRQID=WAITBND (x'82') It is activated through TREADY TASK=
- After the CMT task has written an SCRT89 record, it does SETIME and WAIT.
- The CMT task executes mostly enabled for interrupts. Page faults can occur.
- When the CMT task is started, it does STXIT AB to establish an AB exit.
- In case the AB exit is entered the first time, the CMT task is restarted internally.
- If the AB exit is entered the second time, no internal restart is done. The CMT task must be started manually through
  - // EXEC IJBCMT,PARM='START ID=xxxx'

---

## System Resources

Subpool ICMTSP in System Getvis Storage (both LOC=BELOW and LOC=ANY requests). The storage is freed after the SCRT89 record has been written to disk.

---

## New Macro

SGCMT



---

# Program Retrieval

## External and Internal Interface

The program retrieval provides a set of services either to get the information about an executable program or to load such a program into the storage. The programs are contained in a partitioned data set, the so called LIBRARY. This library is divided into sublibraries each of these may contain programs (or phases). The services are realized by means of supervisor calls.

The SVCs are:

SVC X'01' (FETCH macro)  
SVC X'02' (B-Transient load)  
SVC X'04' (LOAD and SLOAD macros)  
SVC X'05' (A-Transient load)  
SVC X'17' retrieves load address; req. can only be JCL or B-trans.  
SVC X'30' (C-Transient load)  
SVC X'33' (HIPROG macro)

The interface to these SVCs is described in "Supervisor Call Interrupt (SVC)" on page 27. Any of the above SVC routines has a common interface to the program retrieval service, the so called FETCH / LOAD service. This interface is described below:

*Input:*

Register 1 = address (parameter-list | phasename)

Register 0 = null  
| address(loadpoint) for SVC X'02',X'04',X'05',X'30',X'41'  
| address(entrypoint) for SVC X'01'  
| address(area, where loadpoint should be stored, is passed  
for SVC X'17')

parameter list = [id,addr(phasename),flag,addr(local-list)]

id = 00 - for normal LOAD / FETCH

01 - ICCF load request

02 - CDLOAD load request

03 - SLOAD request

04 - reserved

flag = 80 - return code requested

40 - SVA load / update

20 - no SDL search

10 - reserved

08 - directory entry with SDL format

04 - system search sequence

02 - directory entry

01 - bypass program fetch (phase in SVA or TXT=NO)

addr(local-list) = address(list) | null

Register 2 = addr (comreg) of pseudo partition if identified as ICCF request.

*Output:*

If successful, requested directory information and / or phase processing.

Register 0	NIL for SVC X'17'
	Address of entrypoint otherwise
Register 1	NIL for SVC X'17'
	null
	address of directory entry in local list
Register 2	NIL for SVC X'17'
	Address of entrypoint (otherwise)
	Return code if requested

**Note:** A load point must be specified for self-relocatable phases.

**RMODE Considerations**

All parameter lists passed to program retrieval must be located below 16MB.

**31-Bit Considerations**

Program Retrieval executes mostly in AMODE 24 and switches to AMODE 31 only when necessary, for example, to load a phase above 16MB, to do the relocation above 16MB, to call subroutines requiring AMODE 31....



## Structure of the FETCH Environment

The diagram in Figure 134 gives an overview of the flow of control for the execution of a FETCH request.

Part 1 shows the actual control flow, part 2 shows the interrelationship between logic and control blocks.

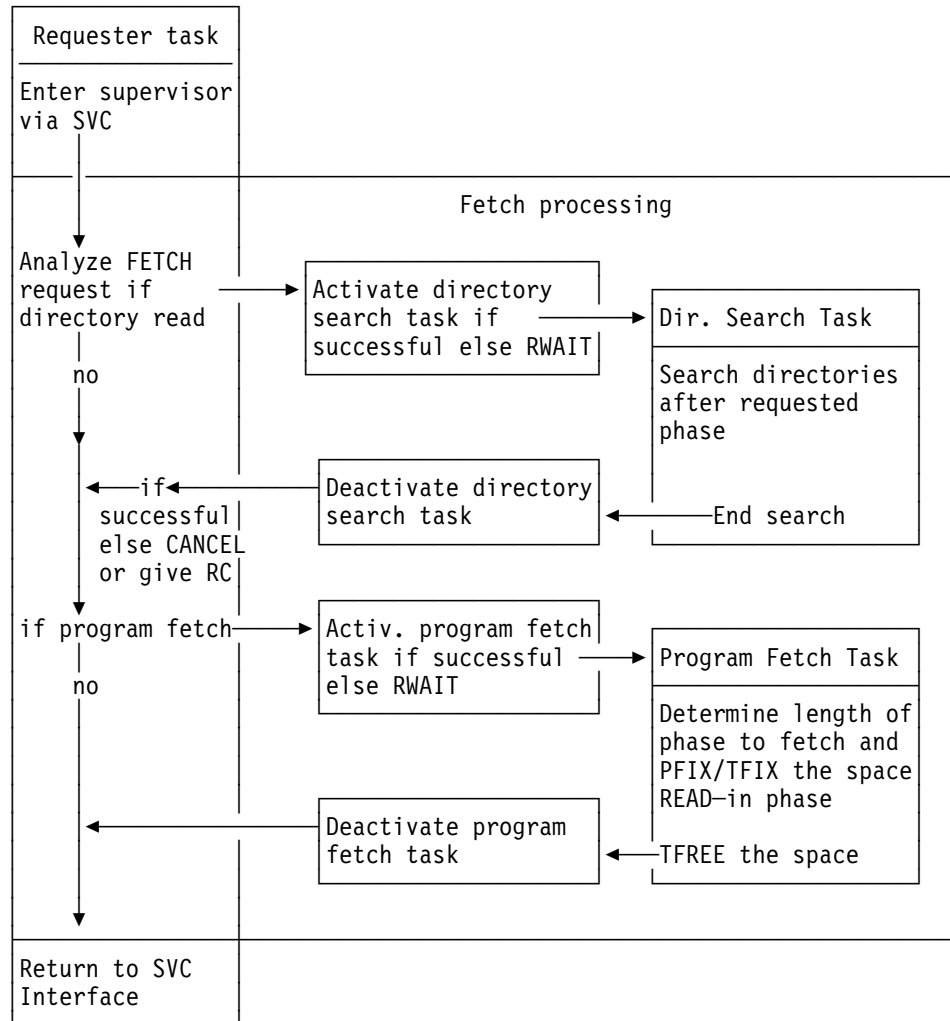


Figure 134 (Part 1 of 2). Fetch Control Flow

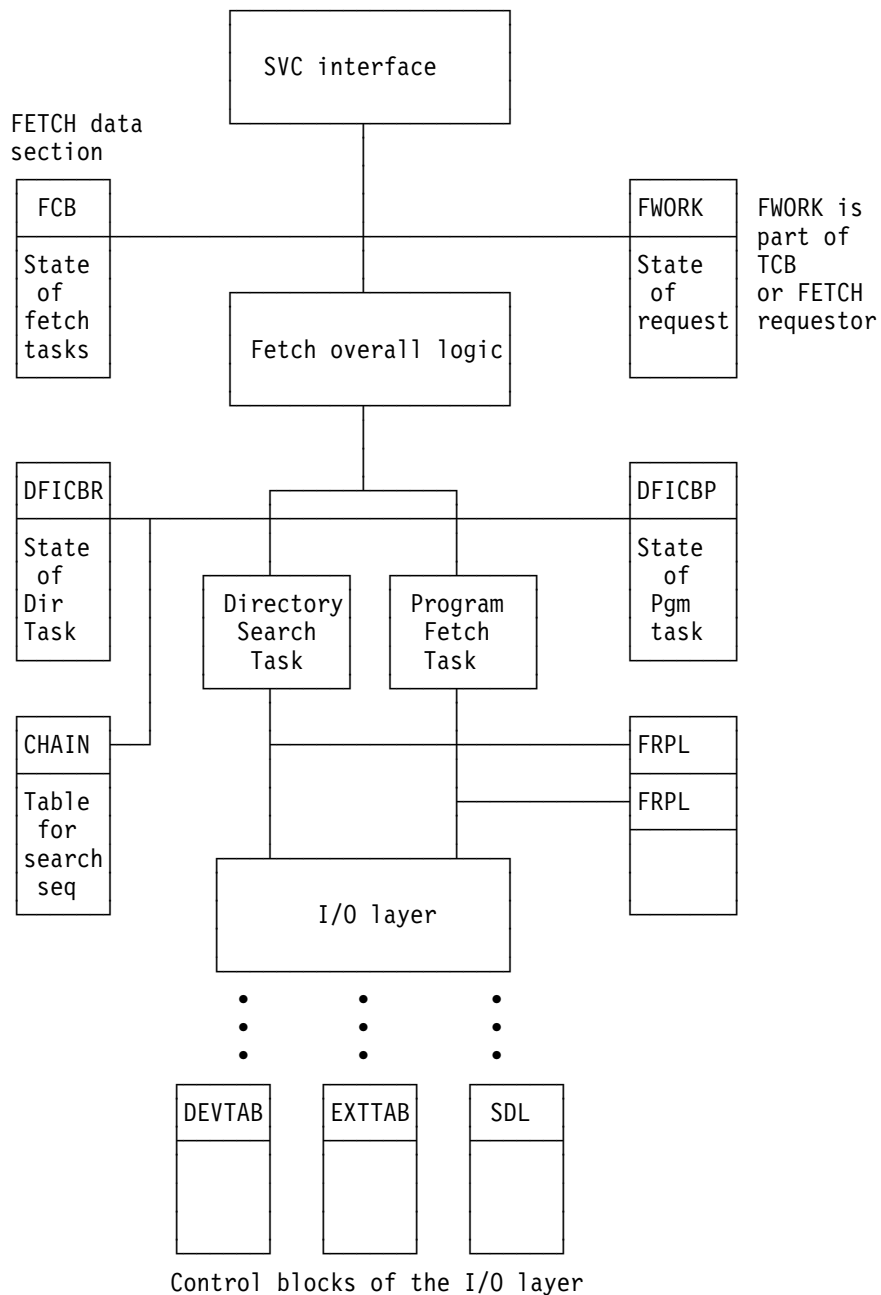


Figure 134 (Part 2 of 2). Fetch Control Flow

## Fetch Concept in Librarian

The librarian supports a uniform and condense-free library concept. A Library (NLIB) consists of a non-empty set of sublibraries each may contain members of various types like PHASE, MODULE, PROCEDURE etc. A sublibrary consists of a directory, alphanumerically ordered after 'TYPE.MEMBERNAME', and a member space. It may have more than one extent on more than one volume of the same disk device type. For faster search algorithm, the directory can be accessed via an index set (B-tree).

The physical organization of the library is done into so called Library Blocks (LBs) of the size of 1KB. The LBs are comparable to the CIs (Control Intervals) in VSAM. A LB contains the data record and VSAM like control information. This is called LBCF and consists of CIDF (Control Interval Definition Field), RDF (Record Definition Field), phase ID and LB chaining field. The next logical LB entity is addressed by the LB chaining field. In such a way the requirement of condense-freeness is satisfied.

As a consequence however, the contiguity of the directory and the space of an individual member cannot be guaranteed. In a frequently updated library respectively sublibrary the degree of fragmentation (directory-, index- and member-space) is increased during its lifetime. The resulting FETCH performance will be essentially decreased. A reorganization of the library is recommended for a proper FETCH performance.

For CKD devices the search on key high or equal is no longer used.

The system library IJSYSRS supports only one extent (on a single volume) and contains at least one sublibrary called SYSLIB. The system library starts on a fixed disk location and contains at least all phases and procedures necessary for IPL.

The library-sublibrary pairs, active in the system, are described by control blocks located in the System GETVIS area. The allocation of these pairs to the VSE partitions is given in the Library Offset Table (LOT).

## Librarian Structure

### *Library Format*

The following figure shows the structure of the LIBRARIAN in such a detail necessary for understanding the FETCH / LOAD processing.

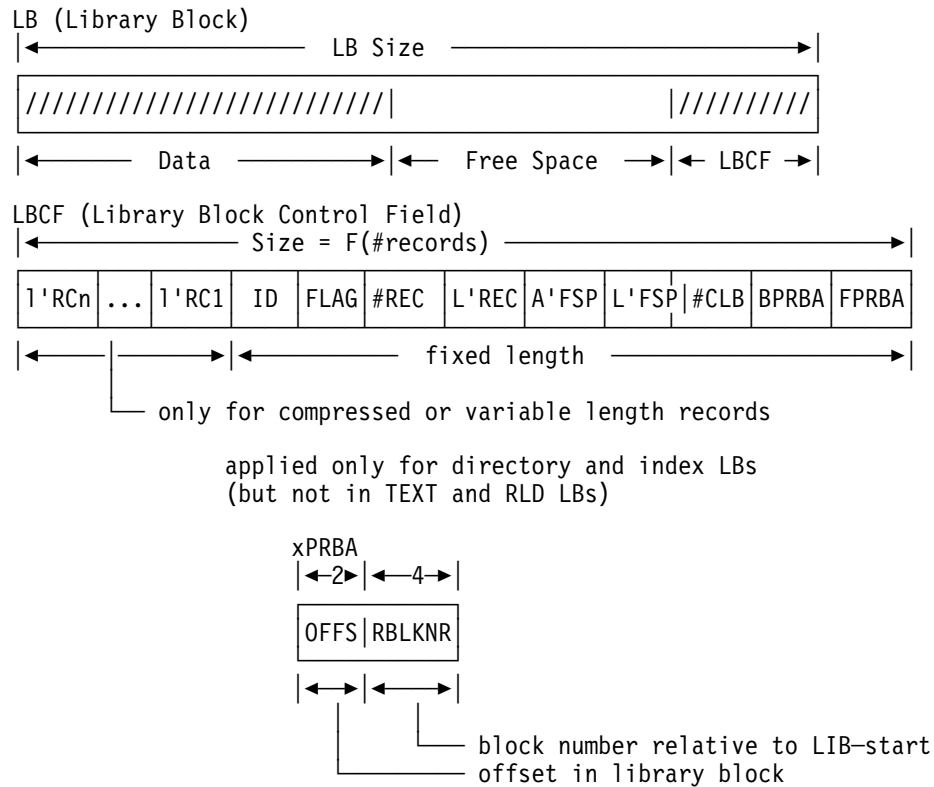


Figure 135. Library Format

The abbreviations are:

- F(#records): function of number of records contained in the LB
- l'RCn: length of record number n (at least one record differs in length from the others)
- ID: phase ID
- #REC: number of records
- L'REC: length of records (if all records of same length)
- A'FSP: begin address of free space
- L'FSP: length of free space
- #CLB: number of contiguous LBs following this LB
- BPRBA: backward pointer RBA (relative byte and block address)
- FPRBA: forward pointer RBA of next logical LB
- xPRBA: FPRBA or BPRBA

## LIBRARY STRUCTURE

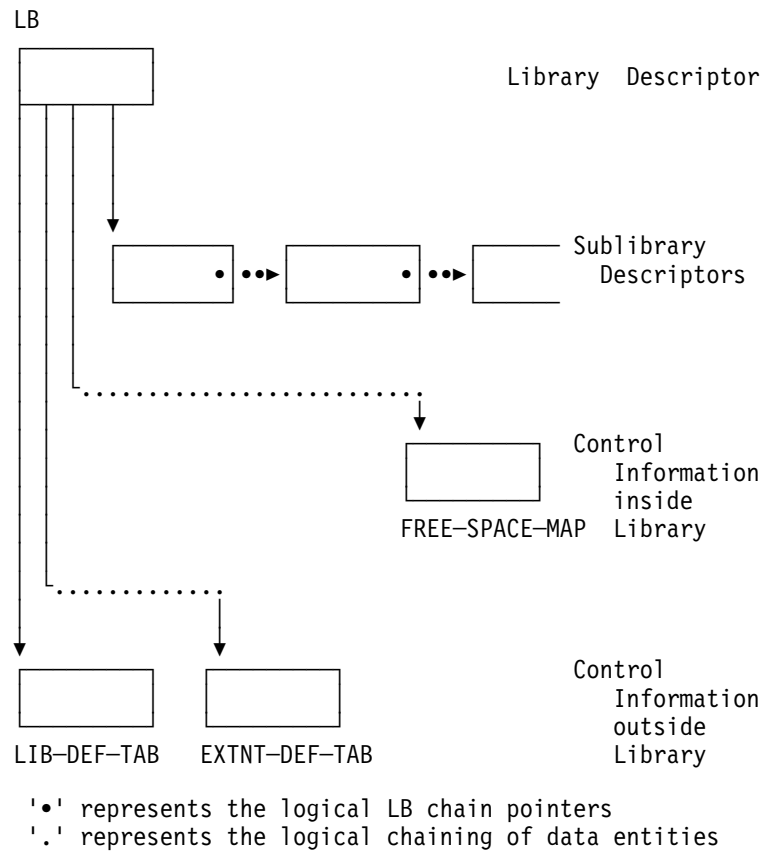
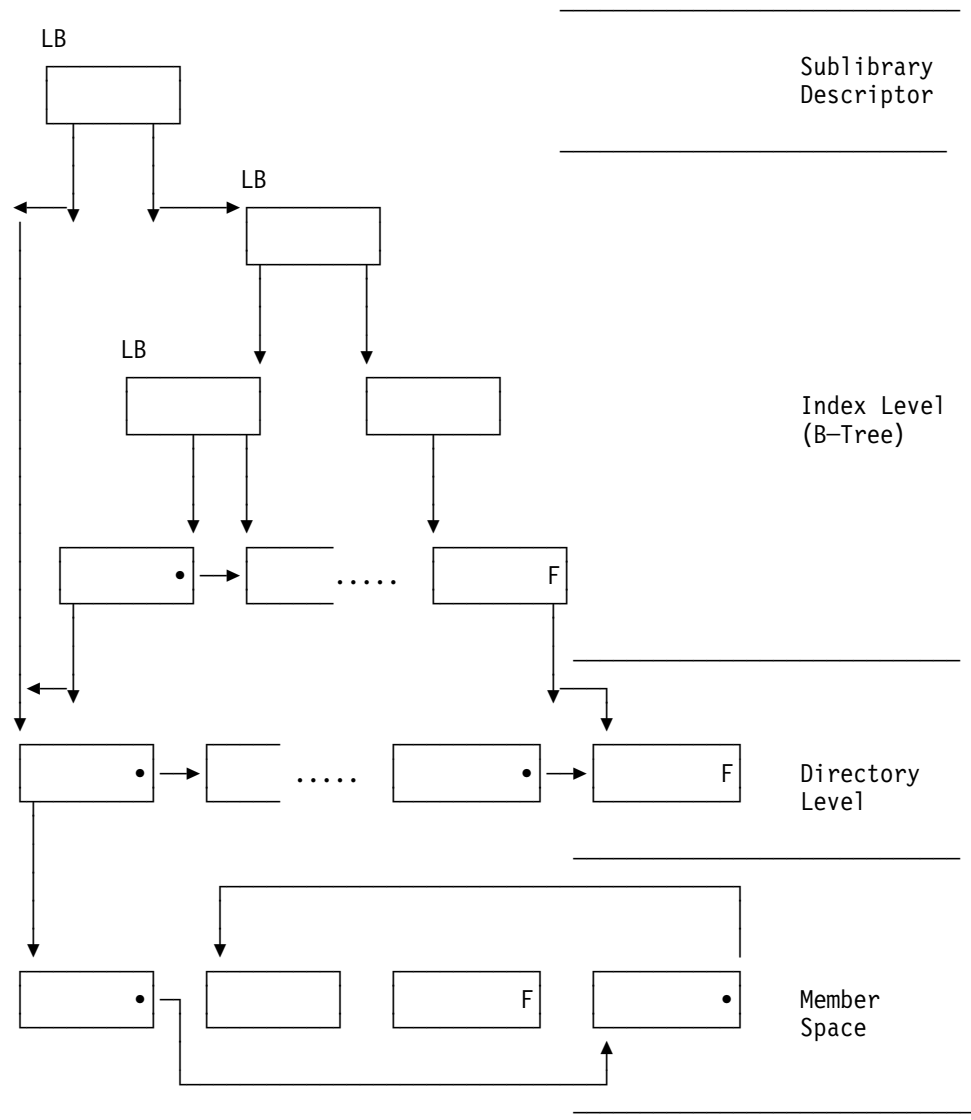


Figure 136. Library Structure

### Notes:

1. The control information tables are not necessarily located as physical fields in the library. They may be built during "Library Allocation" time by means of label information etc...
2. All directory LBs are on the same (lowest) index level and are alphanumerically sorted after "TYPE.MEMBERNAME". The highest index level of a sublibrary consists of one or more LBs (performance considerations).
3. The data length of TXT, or RLD LBs is L'LB - L'LBCF.
4. The EOB indication for DIR or INDEX LBs is given by:  
LBCF.L'REC = X'0'
5. End of a logical chain (for example, member, directory ) is given by:  
FBRBA = X'FFFFFFFFFFFFFF'.

SUBLIBRARY STRUCTURE



'•' represents the logical LB chain pointers  
 '.' represents the logical chaining of data entities

Figure 137. Sublibrary Structure

### Directory and Index

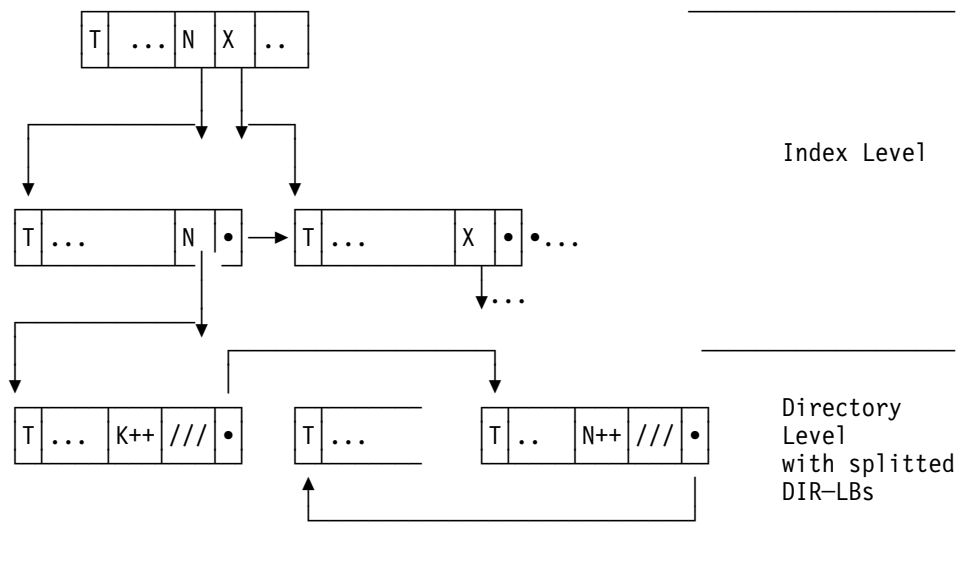
Each member of a sublibrary is described by a corresponding directory entry. Directory entries on one physical LB are accessible via an index entry in the (next higher) index level. If this index level consists of more than 3 LBs, then a higher index level is provided in order to support a fast search algorithm.

However, at any point in time these relationships might not be valid: a LB-split of a lower level LB can be already successfully performed but is not yet reflected in the higher level index-LB.

In such a case more than one I/O operations must be done for the same index level.

The data part of a directory or index LB may be empty.

As a consequence, the SLD might not be consistent to the directory LBs, therefore the possible LB-split must be considered by the directory search algorithm too.



**Note:**

- T: TYPE entry
- N: Index entry
- X: Index entry
- K++: Directory entry
- N++: Directory entry

- '•' represents the logical LB chain pointers
- '.' represents the logical chaining of data entities

Figure 138. Directory and Index

The general format of a directory is as follows:

```
DIRECTORY      : header      : descriptor record
                lb-list     : ◀ dir-LB ▶
dir-LB         : datalist   : data1 | data2 | NIL
                lengthlist : length-data | NIL
                LBCF        : control field

data1          : datah1     : ◀type-entry▶ v ◀index-entries▶
                datarest1  : data1 | NIL

data2          : datah2     : ◀type-entry▶ v ◀dir-entries▶
                datarest2  : data2 | NIL

length-data    : ◀ tail (length-data), head (data i) ▶

typ-entry      : typename   : (PHASE,PROCEDURE,...)
                typflag    : flag value
                typdata     : type data

dir-entry      : dirname    : name
                dirflag    : flag value
                dirdata     : directory data

Data invariant : tail(datah1) not= NIL
                tail(datah2) not= NIL
```



### Library Member

A member is the smallest unit of data which is accessed by the FETCH services. A member of the type=PHASE uses the complete data section available on the LB. A member starts always on LB boundary and consists of two different types of information:

TXT: Contains the executable code is cataloged by the Linkage Editor.

RLD: Contains addresses in the TXT to be relocated.

The following diagram shows the relationship between directory entry and member:

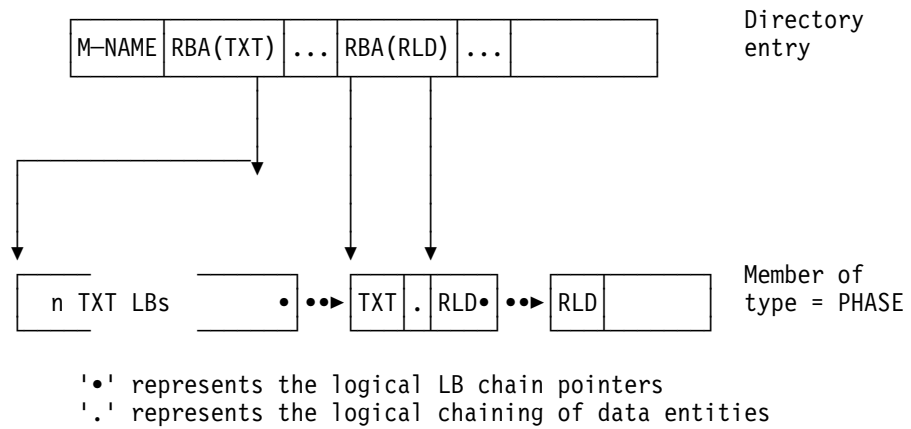


Figure 139. Library Member

The general format of LB of a PHASE - member is as follows:

```

PHASE      : PHASELIST : <LB-PHASE>

LB-PHASE   : DATA      : record
            : LBCF       : control field

RECORD     : TXT        : phasetxt | NIL
            : RLD        : <rlditems> | NIL

Data invariant : RECORD not= NIL
  
```

## Shared Virtual Area (SVA)

The SVA (24-bit) (Figure 5 on page 12) contains

- A system directory list (SDL) providing a list of either descriptors of programs (phases) located in the SVA or in-storage directory entries of highly used programs (phases) located in the SYSLIB sublibrary of the SYSRES file. It contains both the entries for the SVA (24-bit) and the SVA (31-bit).
  - The SDL entry is a subset of the directory entry of the library and contains all information required to satisfy the fetch / load services. The SDL has fixed-length entries of 72 bytes. The last entry contains 8X'FF' as phasename. The external directory format is mapped into an internal directory format which is also used as SDL entry format.
- Highly used programs (phases) located in the SVA can be shared between partitions (virtual library). These programs run with the PSW of the requesting task. SVA resident programs must be relocatable and refreshable. If used in connection with Fast B- and C-transient Fetch (Move-mode), the SVA resident transients must be self-relocatable. In any case, the programs (phases) must be loaded into the virtual library during IPL or job control time. Any subsequent Fetch request for a B- or C-transient moves the SVA copy of the phase into the LTA/CRT area, instead of loading it from the library on disk. A phase is loaded in the SVA by a SLOAD request issued by the librarian. Depending on the RMODE attribute of the phase, the librarian passes a loadpoint pointing to the SVA (24-bit) or SVA (31-bit).

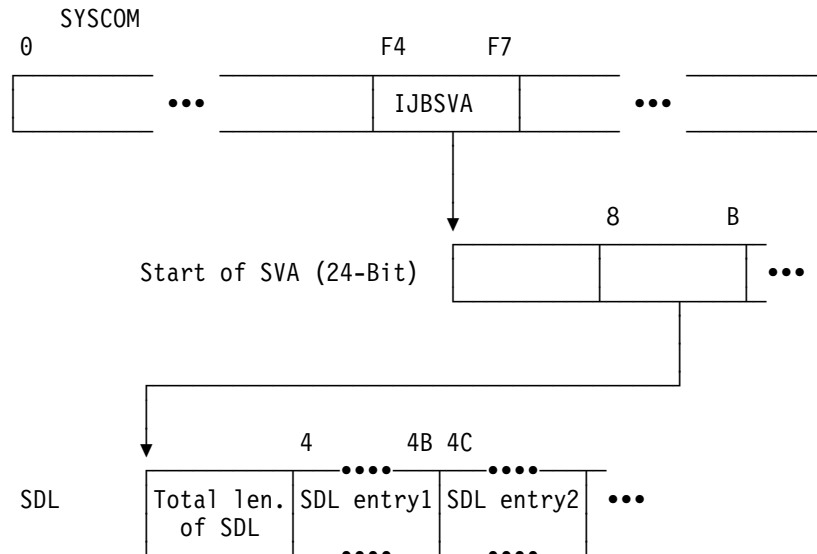


Figure 140. How to Locate SDL Entries

DEC	HEX	Label	Description
0	0	SDLESEG1	Directory Entry (DE) – common segment
0	0	SDLENAM	Member name
8	8		Reserved
9	9	SDLEDEF1	Attributes for DE (flag byte)
		SDLEETYP	X'80' Type of entry = type
		SDLEEHLX	40 Type of entry = high level index
		SDLEEDIR	20 Type of entry = directory
			10 – 01 Reserved
10	A	SDLEPRBA	PRBA of member
16	10	SDLECONT	Number of contiguous LBs
18	12		Reserved
20	14	SDLEPFL	User area1 (type = PHASE)
20	14	SDLEFLG	Flags
		SDLEBSR	X'80' Self relocating phase
		SDLEBRL	40 Relocating phase
		SDLEBSE	20 SVA eligible
		SDLEBSV	10 Phase in SVA
		SDLEBPC	08 PCIL flag for incore directory
		SDLEBNF	04 Not found flag (incore directory)
		SDLEBAC	02 Entry active (incore directory)
		SDLESVPF	01 SVA eligible and PFI request
21	15	SDLESWT	Switches
		SDLECLM	X'80' Set SDL: move mode phase
		SDLECLS	40 Set SDL: SVA eligible
		SDLERMOD	20 RMODE: 1=ANY, 0=24
		SDLEAM31	10 AMODE: 10=31, 11=ANY
		SDLEAM24	08 00=24, 01=24
			04 – 01 Reserved
22	16		Reserved
24	18	SDLEPLN	Length of phase(TXT) in bytes
28	1C	SDLELPT	Load point at link-edit time
32	20	SDLEENP	Entry point at link-edit time
36	24	SDLESTR	Partition start at link-edit time
40	28	SDLERLD	Number of RLD items
42	2A	SDLERLDA	PRBA of first RLD item if any, otherwise x'FF'
48	30		Reserved
56	38	SDLESVAP	Entry point in SVA if any, otherwise X'00'
60	3C	SDLEIDEN	Library block id
64	40	SDLEALIB	Address of LIB-DEF-TAB
68	44	SDLEASLB	Address of SUBLIB-DEF-TAB
72	48		Total length

Figure 141. SDL Format of a Directory Entry

A program is loaded into the requesting partition only, if it is not in the virtual library.

A phase is loaded into the SVA at the next available double-word boundary.

## Directory List Support

Directory list support allows the user to create in-storage directories of highly used phases. Once initialized, loads and fetches of such selected phases are made without searching the allocated sublibrary directories on disk. A system directory list, available to all partitions, is provided in the SVA for phases resident in the SVA and for other highly used phases.

Local directory lists may be created by the user at any time. A local directory list exists for the duration of the job step, in which it is created.

It should be noted that an in-storage directory entry in the user's partition does not contain any valid information, except for the phasename, length of directory entry and entry status, until the first FETCH or LOAD request for the phase specifying this entry has been executed. The first FETCH or LOAD request for the phase activates the entry and subsequent requests can use this entry.

If an in-storage directory entry points to a phase which is already deleted, then FETCH reacts as if the 'phase not found' condition had occurred - that means: the phase will not be loaded. Notice, that in previous releases the phase was loaded in this case.

The user macros LOAD, FETCH and GENL generate the new directory entry format if the option DE=VSE is specified. Otherwise a list in old DE-format is generated.

Both the old DE=YES and the DE=VSE version of LOAD, FETCH, and GENL macros are supported by the FETCH environment.

But in regard to LIBRARIAN and security aspects, some directory entry fields of the DE=YES format are no longer supported or their meaning has been changed.

DEC	HEX	Label	Description
0	0	DIRNAME	Member name
8	8		*** internally used ***
11	B	DIRN	Number of halfword containing user data
12	C	DIRTT	Number of TXT blocks (1024 bytes)
14	E	DIRNN	TXT bytes in last TXT block
16	10	DIRC	Flags
		SELFREL	X'80' selfrelocatable
		RELPHASE	X'40' relocatable
		SVAELIG	X'20' SVA eligible
		SVAPHASE	X'10' phase is SVA-loaded
		PCIL	X'08' not-SYSLIB flag for in-core-DE
		NOTFND	X'04' not found flag
		ACTIVE	X'02' active DE (but possibly not found)
			X'01' reserved
17	11	DIRSWIT	More flags
		DIRJCLM	X'80' MOVE MODE flag from SET SDL
		DIRJCLS	X'40' SVA eligible
		DIRRMOD	X'20' RMODE: 1=ANY, 0=24
		DIRAM31	X'10' AMODE: 10=31, 11=ANY
		DIRAM24	X'08' AMODE: 00=24, 01=24
18	12	DIRPPP	Loadpoint at LINKEDT time
21	15	DIREEE	Entrypoint at LINKEDT time
24	18		*** not supported ***
27	1B	DIRAAA	Partition begin at LINKEDT tme
30	1E	DIRVEE	SVA entry point (if SVA-loaded)
34	22	A	*** not supported ***
38	26		Total Length

Figure 142. Layout of the Old LIBRARIAN User DE-Format (DE=YES)

DEC	HEX	Label	Description
0	0	DIRNAME	Member name
8	8		X'FFFFFF'
11	B	DIRN	Number of halfword containing User data (X'0E')
12	C	DIRLMBR	Length of phase in bytes
16	10	DIRC	Flags
		SELFREL	X'80' selfrelocatable
		RELPHASE	X'40' relocatable
		SVAELIG	X'20' SVA eligible
		SVAPHASE	X'10' phase is SVA-loaded
		PCIL	X'08' not-SYSLIB flag for in-core-DE
		NOTFND	X'04' not found flag
		ACTIVE	X'02' active DE (but possibly not found)
			X'01' reserved
17	11	DIRSWIT	More flags
		DIRJCLM	X'80' MOVE MODE flag from SET SDL
		DIRJCLS	X'40' SVA eligible
		DIRRMOD	X'20' RMODE: 1=ANY, 0=24
		DIRAM31	X'10' AMODE: 10=31, 11=ANY
		DIRAM24	X'08' AMODE: 00=24, 01=24
18	12		Reserved
20	14	DIRACOPY	P T R T O D E - C O P Y
24	18	DIRALPT	Loadpoint at LINKEDT time
28	1C	DIRAEPT	Entrypoint at LINKEDT time
32	20	DIRAPART	Partition begin at LINKEDT time
36	24	DIRASVA	SVA entry point (if SVA-loaded)
40	28		Total Length

Figure 143. Layout of the LIBRARIAN User DE-Format (DE=VSE)

The length DIRN is given in number of halfwords following this field. If the user does not specify the length (field is zero), nothing is moved into the user's directory entry.

## Fetch Initialization

Before a FETCH service can be activated, all physical and logical descriptions about the library (-ies) must be available. Especially the control blocks for the SYSLIB (SYSRES) must be initialized before the first FETCH request can be satisfied.

These control blocks are:

- DEVTAB
- EXTTAB (one entry only because SYSLIB consists of one extent)
- LIBRARY DEFINITION TABLE for the IJSYSRS file
- SUBLIBRARY DEFINITION TABLE for SYSLIB sublibrary

*Functions and Algorithms*

The control blocks and their related functions are as follows:

```
EXTTAB ==
  init1(SYSCOM,GETVCE(IJSYSRS),SYSLIB-PUB)
```

```
DEVTAB ==
  init2(EXTTAB(SYSLIB),SYSLIB-PUB)
```

The access path to index set is as follows:

```
start(IJSYSRS)---RBA --->Lib-Descr(IJSYSRS)
                ---ptr --->Slib-Descr(SYSLIB)
                ---PRBA--->Index-Set
```

The relationships between the control blocks are the same as for the LIBRARIAN. A so called system searching chain is established during the FETCH initialization and will be maintained by the LIBRARIAN services.

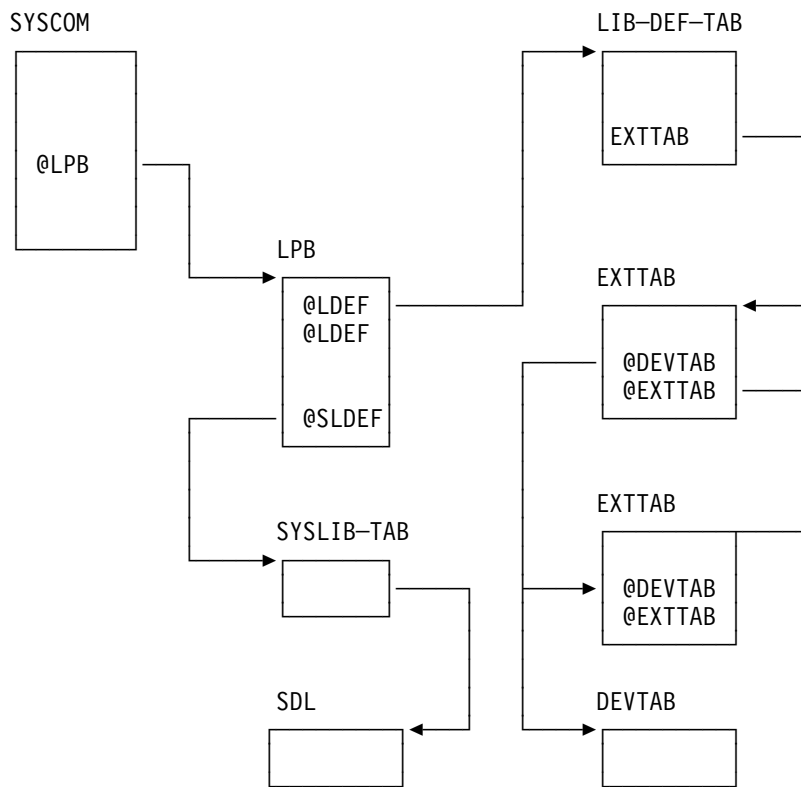


Figure 144. Relationship Between Library Control Blocks

**Notes:**

1. The SLD of the SYSLIB is built by the librarian at end of IPL time.
2. The meanings of the various control blocks are given below.

## FETCH/LOAD Processing

The SVC interface routine at SGCFCH passes control to the fetch overall logic routine in SGDFCH. Before entering the fetch routine, the return address is stored in register 11. Moreover, in register 9, a parameter is stored indicating the SVC interface routine that requested the fetch routine. The meaning is as follows:

- 0 = Requested by SVC X'17'.
- 4 = Requested by any other routine.
- 8 = Requested by SVC X'33'.

Registers 8 through 14 are saved in the requestor's TCB. The user supplied register 1 points either to a parameter list or to an entry of the fetch/load list or a phasename (each time an 8-byte area). For a more detailed description please see the section 'External and Internal Interface'.

### Directory Searching Sequence and Directory Entry Processing

The directory search is performed by the directory search task. Prior to accessing the directories, the searching sequence must be determined. The searching sequence depends on the following conditions:

1. Request given by attention task
  - a. SDL (system directory list)
  - b. SYSLIB directory
2. Request given in test mode (COMREG byte 59, bit 5=1)
  - a. VIO-directory, if any
  - b. Temporarily chained sublibrary directories (SDL)
  - c. Permanently chained sublibrary directories
  - d. SYSLIB directory
3. No test mode and (\$-phase or SYS=YES)
  - a. SDL
  - b. SYSLIB directory
  - c. Temporarily chained sublibrary directories
  - d. Permanently chained sublibrary directories
  - e. VIO directory, if any
4. No test mode and non-\$-phase and SYS=NO
  - a. VIO directory, if any
  - b. SDL
  - c. Temporarily chained sublibrary directories
  - d. Permanently chained sublibrary directories
  - e. SYSLIB directory

However, a directory search is not necessary if the user has provided an active directory entry for the requested phase. Such a directory entry has been built as a result of a preceding FETCH/LOAD request. It can be provided in one of the following ways:

- As a directory element to which the phase name parameter is pointing (DE=YES in the FETCH/LOAD macro).
- As a directory entry in a local directory list (LIST parameter in FETCH/LOAD macro).
- As an SDL entry (for special system services).



**Note:** It is an essential prerequisite that the FETCH must not be locked against LIBRARIAN services.

If the user has passed a local list, this list is validated and searched for the requested phasename.

If no active directory entry has been provided, the directory search task is activated and the first level chain (FETCH CHAIN) is built.

The so called first level entries are available for SDL and SYSLIB, whereas so-called second level entries are reserved for the concatenation chain. In the later case, the addresses of the Library Definition Table (LDT) and the Sublibrary Definition Table (SDT) are calculated by means of the actual entry in the LOT (Library Offset Table).

The directory search operates on a set of control blocks described as follows:

- DEVTAB (Device Definition Table)  
The DEVTAB describes the library device in all its physical aspects, such as device types and device characteristics.
- EXTTAB (Extent Definition Table)  
The EXTTAB describes the location of the library on a device and provides the relation to the RBA addresses. Moreover, it contains the PUB index of the device.
- LPB (Library Pointer Block)  
The LPB is the focal point for any access to chained libraries. It provides the maximum number of entries in the search chain (=maximum number of chained sublibraries) and addresses to the searching chains of library-sublibrary pairs. There is a temporary S-chain (search chain) which is reset at EOJ time and a permanent S-chain which must explicitly be reset. The existence of such a search chain is considered by the searching algorithm. The LPB is addressable via SYSCOM. The address pointer will be negative (X'80000000'), if the control tables are not yet initialized.
- LANC (Libdef Anchor Table)  
The Libdef Anchor Table contains for each partition the pointers to the partition-related LOTs (temporary and permanent) within the LOT pool. Each partition is represented by an entry in the LANC table. The partition related entry can be addressed by using the PIK.
- LOTxxxx (Library Offset Table)  
The LOTs (one for a permanently assigned library chain and one for a temporarily assigned library chain) describes the various S-chains of a specific library type in the various partitions. The fields relevant for FETCH are:
- VIO library  
The VIO library has no separate description. Essentially it is identified by its related VIORB. An address to the VIORB is given by a special LOT row.  
  
The FETCH searching algorithm works on an internal control table which is built for each FETCH request. All information of the searching chains in the LOTs is mapped into this internal table. Thus, the complete searching mechanism is staged in three levels:
- The FETCH chain table DSRCHNx located in the fixed part of the supervisor reflects the searching chain described above. The entries for SDL and SYSLIB are filled, while the other entries are dummy entries only. By this way any

unnecessary page fault is avoided if the phase is found in the SDL or on the SYSLIB (\$-phase) The essential information are the address of the DEVTAB and EXTTAB.

- The searching chain of the LOT is accessed whenever an entry in the DSRCHNx is found indicating permanent or temporary chain or VIO-library. If the chain entry is active and the DSRCHNx is not yet initialized, the related LOT is accessed and the first chain entry is taken in order to activate the DSRCHNx entry. otherwise the next chain entry will be taken as long as there are active entries. In the case of end of chain the next DSRCHNx entry will be processed.
- The DEVTAB and EXTTAB entries are required to read on the physical library device.

A FRPL for directory (DIR) respectively VIO read must be set up for each search of a sublibrary. Moreover, the related addresses of the LDT and SDT for DIR-read respectively of the VIORB for VIO-read must be provided in FCHWORK.

If finally the requested phase is found its directory information is built up in the FCHWORK for further processing.

**Note:** FCHWORK is part of the requester's TCB. If the directory entry is found in the SDL and the corresponding phase resides in the SVA, no further processing is done. The entry point address, available in the SDL entry, is passed to the user.

If unsuccessful, the user is notified by a 'not found' indication in the directory entry or by a return code in general register 15 (as RET=YES has been specified) or is canceled with the message 'phase(name) not found'.

After a successful search, the user provided directory entry will be activated and updated.

## Functions and Algorithms

The directory search is structured into two levels, a logical level determined by the searching chain and a physical level for the I/O operations. On the logical side the related control blocks are FCHWORK and FETCH-CHAIN; on the physical side DEVTAB, EXTTAB and SLD are concerned.

The directory search mechanism is provided by the following control blocks and their related functions:

```
FETCH-CHAIN ==
    bldchain ( PARM-LIST, state )

FETCH-CHAIN (entry) ==
    nxtchain ( LOT-CHAIN )

LB-DIR-ENTRY ==
    scandir ( phasename, FETCH-CHAIN )

SDL-DIR-ENTRY ==
    binsrch ( SDL )

FCHWORK == bldwrk ( .-DIR-ENTRY )
```

**Note:** The functions BLDCHAIN and NXTCHAIN build together with the FETCH-CHAIN control table an abstract data type.

The initialization of the FETCH-CHAIN control table is represented by the following algorithm:

```
bldchain (parm-list,state) ==  
  
select  
  
    when state = ATTENTION-mode  
    then FETCH-CHAIN := (SDL,SYSLIB)  
    when state = TEST-mode  
    then FETCH-CHAIN := (LOT-TEMP or SDL,  
                        LOT-PERM,SYSLIB)  
    when state = SYS-mode  
    then FETCH-CHAIN := (SDL,SYSLIB,LOT-TEMP,  
                        LOT-PERM)  
    when state = USER-mode  
    then FETCH-CHAIN := (SDL,LOT-TEMP,  
                        LOT-PERM,SYSLIB)  
  
endselect;
```

The algorithm for provision of the first / next entry of the LOT-CHAIN is given by the following program:

```
nxtchain (LOT-CHAIN) ==

if entry(FETCH-CHAIN) = EMPTY
  then state.LOT-CHAIN := not EOL
  else
  endif
get-next(LOT-CHAIN)      / may post EOL ...
                        ... if no more valid LOT entry/
if state.LOT-CHAIN = not EOL
  then entry(FETCH-CHAIN) := entry(LOT-CHAIN)
  save-ptr(LOT-CHAIN)    / save addr of current...
                        ... of actual LOT entry      /
  else
  endif;
```

The searching algorithm on the LBs is defined by the following program:

```
scandir(phasename,entry(FETCH-CHAIN)) ==

state.DIR := not EOF
do while state.DIR = not EOF
  DIRREAD (ENTRY(FETCH-CHAIN))
                / EOF if no more dir-LBs      /
  do while state.LB = not EOB or state.DIR = not EOF
    get-next( LB )
                / EOB if LB is empty or processed/
    select
      case phasename = name(LB-entry)
        then DIRENTRY := LB-entry
          state.DIR := FOUND & EOF
      case phasename < name(LB-entry)
        then state.DIR = (not FOUND) & EOF
    endselect
  enddo
enddo;
```

The algorithm for searching the sublibraries is given by the following program:

```
find(phasename,parmlist,state) ==  
  
FETCH-CHAIN := bldchain(parmlist,state)  
do while state.DIR = (not FOUND) or FETCH-CHAIN = EOL  
  get-next(FETCH-CHAIN)  
    / EOL if FETCH-CHAIN is processed/  
  if entry(FETCH-CHAIN) = 2NDLEVEL and FETCH-CHAIN = not EOL  
    then entry(FETCH-CHAIN) := nxtchain(LOT-CHAIN)  
      / FETCH-CHAIN = EOL IF LOT-CHAIN /  
      /                               IS PROCESSED   /  
    else  
  endif  
  if LOT-CHAIN = not EOL  
    then if entry(FETCH-CHAIN) = SDL  
      then DIRENTRY := binsrch(phasename,SDL)  
      else DIRENTRY := scandir(p-name,entry(FETCH-CHAIN))  
        / state.DIR = (not FOUND) if phase not In dir/  
    endif  
  else  
  endif  
enddo;
```

## Program Fetch Service

The program fetch task provides services for:

- Load-in of phases (TXT processing)
- Address relocation (RLD processing)

In opposite to the DIRECTORY SEARCH TASK, the library and the sublibrary are known. The addresses of the related control tables are part of the internal directory entry.

Essentially, the related TXT and RLD LBs must be read in and be processed. To do so, CCW-programs have to be generated. If the storage is virtual, the input space must be fixed before any read request can be performed. If the phase is to read into PFIxed storage, the phase area is completely PFIxed before reading the phase into storage. Otherwise the input space must be TFIxed. The size of TFIxed area is calculated via the number of contiguous TXT-LBs. But the CCWs are generated in dependence of the actual TFIxed space (not all space might be TFIxed) or the number of contiguous TXT-LBs.

For TXT processing the first RLD-LB is read-in with the first TXT-LBs (via chaining of TXT and RLD CCW-programs due to performance reasons).

This can only be done, if RLD and TXT are located on the same DASD device.

The offset of the RLD in a LB is provided in the directory entry (type=phase user information part).

## Program Fetch Interface

The program fetch task operates on a library member. Therefore the library device and the (absolute) disk address must be known (or at least derivable).

The related directory entry is provided in the FCHWORK area before activation of the PROGRAM FETCH task. It contains the relative block numbers of begin of TXT or RLD. The disk device address and the disk address must be calculated by means of DEVTAB and EXTTAB information.

The addresses of the related DEVTAB and EXTTAB are saved in FCHWORK too.

The interface to the I/O layer is the FRPL. As for the Directory search the FRPL must be initialized before the first read request for TXT or RLD LBs can be performed.

**Algorithm for TXT processing:** The algorithm for TXT processing is as follows:

```
gettxt (fchwork) ==

BEG-PHASE := function( d-entry(phase),loadpoint)
LEN-PHASE := function( d-entry(phase) )
END-PHASE := BEG-PHASE + LEN-PHASE
RELO-FACT := function( loadpoint,loadpoint(LINK-EDIT))
validate (BEG-PHASE,END-PHASE)
if RELO-FACT = not 0
  then read(RLD-LB)
  / might result in chaining RLD-CCWs to TXT-CCWs/
  else
endif
if pfix of phase is requested
  then PFIX(BEG-PHASE,END-PHASE)
  else
endif
  BEG-READ := BEG-PHASE
do while LEN-PHASE > 0
  LEN-READ := function (contiguous TXT-LBs)
  do while LEN-READ > 0
    if is-address-space virtual and no-PFIX-done
      / only the contiguous part is TFIXed /
      then TFIX (BEG-READ,LEN-READ)
    else
    endif
    read(TXT-LBs,1'TFIXED space)
    process(RLDs,REL-FACT)
    if is-address-space virtual and space-TFIXed
      then TFREE(BEG-READ,1'TFIXED space)
    else
    endif
    LEN-READ := function(1'TFIXED space or 1'contiguous part)
    BEG-TXT := BEG-TXT + LEN-READ
  enddo
  LEN-PHASE := LEN-PHASE - LEN-READ
enddo;
```

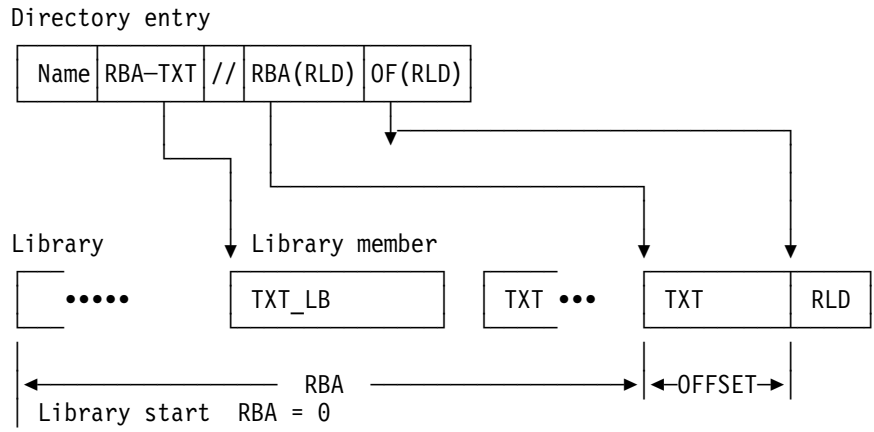


Figure 145. Relationship Between Directory and Phase-Member

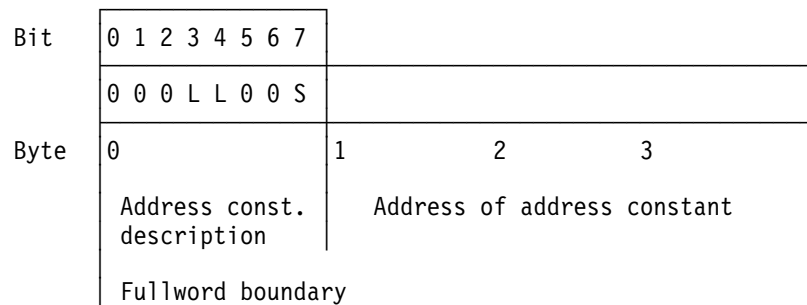
## RLD Processing

The FETCH / LOAD services relocates the address constants given in the TXT part of the requested phase.

The load point of a phase is either provided in general register 0 or is implicitly determined by the load address at linkage edit time. In the later case the load point is the partition start address (behind the save area) plus the difference between load address and partition start address at linkage edit time (given in the directory entry).

The load point of a self-relocatable phase must always explicitly provided.

The relocation of the address constants is performed for relocatable phases. Such a phase contains additional information of the location of address constants, the so called RLD items.



LL = Object length of address constant in TXT  
 11 = length 4, 10 = length 3, 01 = length 2, 00 = length 1  
 S = Relocation factor application  
 0 = Add  
 1 = Subtract

Figure 146. Layout of RLD Items

## I/O Processing

The I/O layer handles all I/O operations for the FETCH/LOAD processing. It provides a control block interface, the so called FRPL. This FRPL must be built for each sequence of I/O operations (like TXT-read-in) and identifies the CCW program to be used, the record and the block-length, the number of records to be read-in and the input area. Its format is described in the section Control Blocks.



## Directory Read Algorithm

The directory read algorithm is given below. The input parameters are the FRPL, the phasename and the (sub-)libraries EXTTAB, DEVTAB and SLD.

```

dirread (FRPL, phasename, sublib) ==

do while FRPLOPC = 1strd
  get-acc (sublib)
  FRPLOPC: = nstrd
  IF sublib.SLD = active and not in back level state
    then call SCANSLD (phasename)
    /searches SLD - returns ok or/
    /
    phase not found in SLD/
  else
  endif
  if sublib.SLD = (inact v in back level state
    v phase not found in SLD)
    then get-LB-addr (index)
    do while index = not processed
      call REQIO / read index-LB /
      get-LB-addr (phasename)
    enddo
  else
  endif
enddo
call REQIO / read directory-LB /
save (LBCF.FRBA); / save addr of next-LB /

```

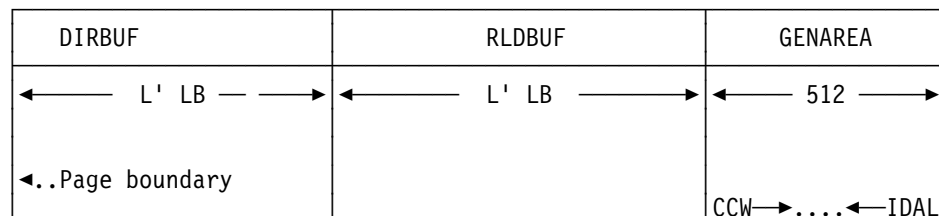
The directory or index LBs are read into the DIRBUF area, part of the pageable supervisor and located on page boundary.

As all other internal FETCH input buffers, the DIRBUF is TFIXED whenever a directory read request must be performed. The related CCWs must be translated.

## TXT and RLD Processing

The I/O of TXT and RLD LBs is performed by a generated CCW program for the TXT LBs and -if appropriate- a command-chained RLD CCW program. The TXT CCWs are generated in a special area, the so called GENarea.

### Layout of the I/O Buffers



**Note:** CCW generation is done upwards  
IDAW generation is done downwards

Figure 147. Layout of I/O Buffers

The algorithm for TXT read-in is given below:

```
txtread(fchwork,FTTAB,EXTTAB)==

do while FRPLOPC = 1strd
  FRPLOPC = nxtrd
  get-LB-addr (phase)
enddo
do while FRPLNRC > 0
  call REQCCW
  / provide space for next CCW /
  do while FRPLNRC > 0 & not EOG
    / EOG = END OF GENERATION /
    generate-CCW (FRPL)
    if IDAL = yes
      then call REQIDAL
      / provide space for idal /
      generate-IDAL(CCW-addr)
    else provide-REAL (CCW-addr)
    endif
  call REQCCW
  FRPLNRC = FRPLNRC - 1
enddo
if last TXT LB processing
  then if len (TXT) < len(LBCIF)
    then if last TXT LB not contiguous
      then call REQLBLK
      /adjust CCWs to read LBCIF(2nd last TXT-LB/
      else call REQFBA
      /do not read LBCIF(2nd last TXT-LB) /
      set-CCW-len (FRPLLRC)
    endif
  else
  endif
  else
endif
if RLD = delayed
  then chain (RLD-CCW)
  else
endif
call REQIO/ perform I/O request /
enddo;
```

## RLD Read

A RLD read request supplies the information necessary to relocate the address constants of the relocatable phase to be fetched or loaded. The RLD-LBs are read into the RLD-buffer, from where the RLD items are processed by the program fetch task.

The necessary data are:

- Start address for the library is available in the EXTTAB.
- Relative start address of the phase is available in the directory entry as a library block number.
- Relative start address of the RLD item.

The first RLD block is read-in with the first TXT blocks (by means of chaining of TXT and RLD CCW programs). For any further RLD LB a separate SVC X'0F' must be issued.

## DASD Sharing Environment

A DASD sharing environment is built of two or more CPUs which are operating on common DASD devices. In general there is no direct signalling between the CPUs. Any data access control must be done via gating the shared DASD devices. The related hardware facilities are the DEVICE-RESERVE and DEVICE-RELEASE commands. That means gating on device level.

The software however wants to provide a locking facility for the entity 'data set'. Therefore a special data set (the LOCK FILE) is established by the software, which describes the locks of all DASD shared resources in the system. Only the device containing the LOCK FILE is protected by the hardware facilities.

The FETCH is concerned by DASD sharing in regard to PHS-LIBs and corresponding SLDs.

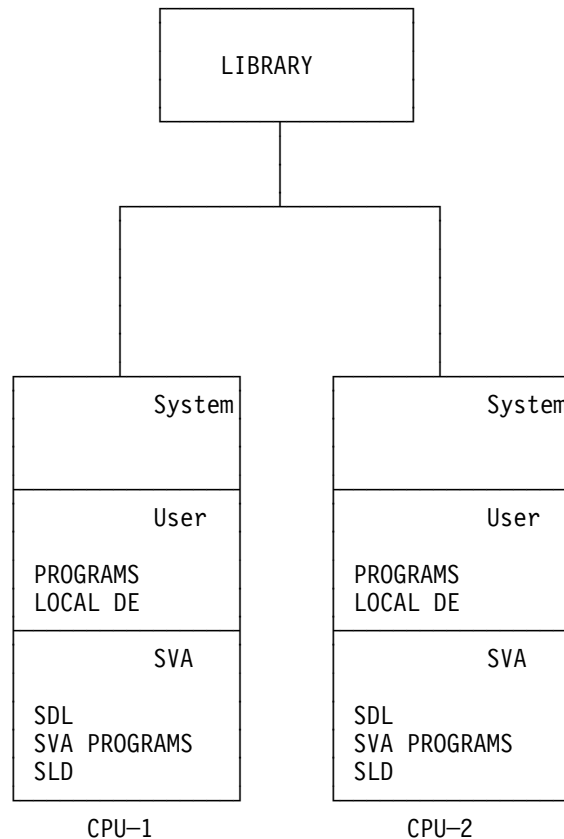


Figure 148. DASD Sharing Environment

## The Second Level Directory (SLD), General Remarks

The SLD was introduced in order to have a quick access to a directory entry.

It has one entry for each directory block of a sublibrary. The entry contains the highest phase name, for which a directory entry is contained inside the directory block and the relative block address. So, by searching through the SLD, Fetch can find at once, (that means with only one I/O operation) the directory block which contains a special directory entry.

Highest phase name inside directory block 1	PRBA of directory block 1	• one entry for each directory block
•	•	
•	•	
highest phase name inside directory block n	PRBA of directory block n	

Figure 149. SLD Layout

The SLD of the system sublibrary IJSYSRS.SYSLIB is initialized at IPL time; the SLD of the private sublibraries at LIBDEF time.

## Initiation of a SLD Update

Some operations, for example the deleting, cataloguing or renaming of a phase, change the directory and might leave the SLD in a back level state.

A SLD update should be done to avoid performance degradation. The SLD update is done by the librarian after a delete, catalogue etc. command was given.

If shared disks are used and the delete command for example, was given by another CPU, then it is Fetch which makes the SLD update.

Fetch identifies a back level SLD by the following criterions:

- More than one library block had been read in to find the directory entry although the SLD was used.
- The SLD entry does not point at all to a directory block, but e. g. to a TXT or RLD block (space reclamation took place).

OR

- The library block found by SLD search is flagged as deleted (LBCFFLG2.LBCFMDEL set) and the library is on a shared DASD.

If Fetch identifies a back level SLD, then a SLD update is initiated unless one of the following is true:

- The sublibrary, whose SLD is in a back level state, is part of a temporary (not permanent) search chain.
- The SLD update for the sublibrary is already initiated, but not finished yet.

## SLD Update / Algorithm

The SLD update processing for a sublibrary consists in the following activities:

- Enqueuing the SLD update request into the SLD queue.
- Activating the Service Task.
- Doing the SLD update by the UPDSLDR routine, which is called by the Service Task.
- Dequeuing the SLD update request from the SLD queue.
- Deactivating the Service Task, if there are no more SLD queue entries to be processed.

Each one of these points will be discussed in more detail.

Notice, that the SLD update itself is done by the Service Task and not by Fetch. That means, that the SLD update runs in parallel with Fetch and does not lead to a lower Fetch performance.

### The SLD Queue

The enqueueing of a SLD update request into the SLD queue is done by the ENQSLD routine, the dequeuing by the UPDSLD routine. The elements of the SLD queue are chained together. The first element is pointed to by SLDACT.

The layout is:

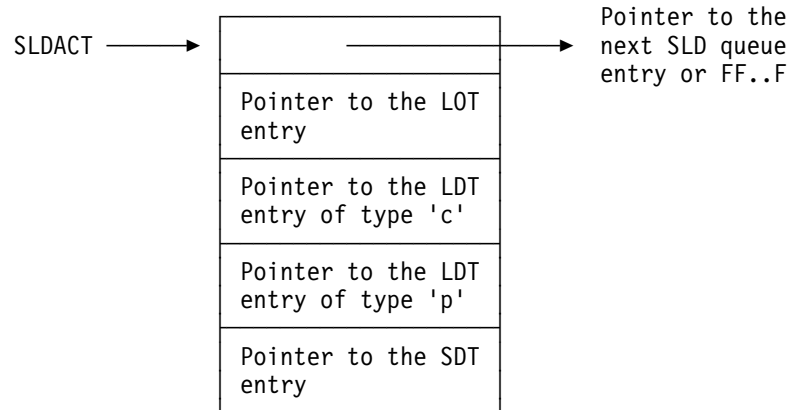


Figure 150. Layout of the SLD Queue

A type 'c' LDT entry is a complete LDT entry while a type 'p' LDT entry refers to a library which is already defined by another LDT. It only contains the library name and a pointer to the corresponding 'c' entry. A type 'p' entry exists when a library is accessed at the same time under another name by the same or another partition.

The layout of an SLD queue entry, disregarding the pointer field, is identical to the layout for the Library Information Area (see *z/VSE Supervisor Diagnosis Reference Librarian*, SC33-6330). Enqueueing a SLD request into the SLD queue means:

- Dequeueing an element from the SLD free chain, pointed to by SLDFREE. The SLD update request is canceled, if there are no free entries in the SLD free chain (SLDFREE=0).
- Putting the right values in the element.
- Enqueueing the element into the SLD queue pointed to by SLDACT.

### The Activation / Deactivation of the Service Task

The activation of the Service Task to do the SLD update is controlled by the flag RQTUPSLD.

If set, the UPDSLD routine is called by the Service Task. The flag is reset if there are no more entries in the SLD queue.

## The UPDSLD Routine

The UPDSLD routine calls the Librarian services INLMSCON, INLMRESN, INLMSLD and INLMDIS to update the SLD.

See *z/VSE Supervisor Diagnosis Reference Librarian*, SC33-6330 for a description of these services.

### *Algorithm of the UPDSLD Routine*

The following actions must be done for each entry of the SLD queue:

- Call INLMSCON to connect the sublibrary.
- Call INLMRESN to get the resource name of the sublibrary.
- Lock the sublibrary.
- Call INLMSLD to update the SLD.
- Unlock the sublibrary.
- Call INLMDIS to disconnect the sublibrary.
- Dequeue SLD entry.

---

## Program Retrieval - Tape Fetch

### General

In a SA environment no library access is done by Program Retrieval. All required phases are on a tape from which they are loaded into storage, part of which is the SVA. Phases are loaded in storage either for immediate execution or into the SVA for later execution. The SVA is used as a library, i.e. it does not contain only sva-eligible phases. After all phases have been downloaded from tape, the tape is no longer used by Program Retrieval. A subsequent load request will cause Program Retrieval to move a phase from the SVA to the specified storage location which can be the partition, LTA or PTA. Of course, SVA eligible phases are still executed in the SVA.

So the Program Retrieval support for the SA environment consists of two parts:

1. read a phase from tape into storage
2. move a phase from the SVA to the specified storage location  
or execute the phase in the SVA.

To move phases from the SVA to the specified storage location the existing Program Retrieval code within the supervisor is adapted.

But there will be a new routine to load phases from tape into storage. This tape fetch routine is needed before the supervisor is loaded. Therefore tape fetch is designed to be part of IPL phases, where it is called with BAL interface, or to be a separate phase which is called by SVC 4 processing after the supervisor has been loaded. The tape fetch routine is implemented as a macro named SATFCH, which is once assembled with the IPL phases \$\$A\$PLBT (first bootstrap phase) and \$\$A\$IPLR (second bootstrap phase) and is once assembled to be a separate phase (\$IJBTFCH). A parameter is used to determine whether the macro is part of the IPL phases (SATFCH CALLER=IPL) or whether it is a separate phase (SATFCH CALLER=SUP).

\$IJBTFCH is loaded by IPL after the supervisor has been loaded using the BAL interface. IPL will put the address of the phase \$IJBTFCH in the SUPAVTEX at label AIJBTFCH. \$IJBTFCH is located on tape after the supervisor phase and is

loaded in storage behind the supervisor. \$IJBTFCH is the last phase which is loaded using the BAL I/F. Afterwards SVC 4 is used (LOAD and SLOAD macro). The tape fetch routine/phase must be self-relocatable.

The SA environment is indicated by a bit SYSCOM.IJBFLG08.IJBSAENV set by IPL. The bit SYSCOM.IJBFLG08.IJBSVALC (sva load complete), set by IPL, determines whether the tape fetch routine \$IJBTFCH or the supervisor code is used to process a SVC 4 request. If the bit is set, the supervisor routine is used, otherwise the tape fetch routine \$IJBTFCH. The indication 'SVA LOAD COMPLETE' (bit IJBSVALC) is only meaningful in a standalone environment. The address of the IPLed tape is set by IPL in SYSCOM.IJBRESDV.

The tape fetch routine will handle both load requests and requests to get directory information. However the tape fetch routine will not use an active DE, passed by the requestor, since it does not contain relevant information for the following TXT processing.

**Note:** The tape fetch routine will support a directory format of DE=VSE and DE=SDL. The DE=YES format is not supported. IPL will change the usage of DE=YES to DE=VSE.

---

## OS/390 Program Retrieval Services

With VSE/ESA 2.4.0 the OS/390 program retrieval services BLDL, LOAD and DELETE have been ported to VSE.

These services are available in OS/390 emulation mode only.

### BLDL Macro

The BLDL macro expands into OS/390 SVC x'12'. It is used to obtain information about library members of type phase. The information is provided in SDL format.

### LOAD Macro

The LOAD macro expands into OS/390 SVC x'08' or SVC x'7A'. It is used to load a phase into virtual storage. Before the phase is loaded, the storage is obtained by means of a GETMAIN-like request, i.e. OS/390 subpools are used.

Normally a phase is loaded in the partition Getvis area:

- subpool 252 if the phase is SVA eligible
- subpool 251 otherwise

Authorized requests can specify the GLOBAL parameter. The phase is then loaded in the system Getvis area.

- subpool 228 if the phase is to be fixed
- subpool 241 otherwise

A phase, that is loaded by the OS/390 LOAD macro is owned by the task, not by the partition. It can be deleted by the DELETE macro. The DELETE must be done by the same task that did the LOAD.

At task termination, all phases owned by a task are deleted.



## DELETE Macro

The DELETE macro cancels a previous LOAD request. The load count of a phase is decreased. When the load count is zero, the storage occupied by the phase is freed, using a FREEMAIN-like interface.

## DCB Parameter in VSE

The DCB parameter is used to specify a task oriented library/sublibrary chain. It is implemented in VSE by applying the Librarian API.

1. define the librarian control block LDCB
  - LDCB LIBRDCB FUNC=GEN,AREA=LDCBAREA
2. establish task oriented library chain
  - LIBRM LIBDEF,LDCB=LDCBAREA,CHAIN=x1,CHAINID=x2  
where x1 specifies up to 15 library/sublibraries and  
x2 specifies the name of the chain.
3. using the dcb parameter
  - LOAD ..,DCB=LDCB

## Directory Searching Sequence

The directory searching sequence for BLDL and LOAD depends on the DCB parameter.

- DCB not specified (DCB = 0)
  - job chain (temporary chain)
  - partition chain (permanent chain)
  - SDL, IJSYSRS.SYSLIB
- valid DCB specified
  - library/sublibrary chain as specified in the DCB
  - SDL, IJSYSRS.SYSLIB
- DCB=1
  - job chain (temporary chain)
  - partition chain (permanent chain)
  - SDL, IJSYSRS.SYSLIB
  - -> DCB=1 is special processing for CICS.

## Implementation

The approach is to use existing VSE functions and put OS/390 dependencies into a new supervisor macro named SGXFCH. This macro contains

- entry points for the BLDL, LOAD and DELETE SVCs
- a BAL interface for ATTACHX processing
  - during ATTACHX processing a phase is to be loaded
- a BAL interface for FREEVIS ALL processing
  - during FREEVIS ALL processing of a task, all phases owned by the task are to be deleted.

On entry, the parameters are checked and then the OS/390 request is mapped to a VSE request. Afterwards existing VSE load/delete functions are called. After the VSE function is completed, the VSE return codes are mapped to a OS/390 return or cancel code.



---

## Machine Check, Channel Check and CRW Handling

Machine checks, channel checks and channel report words (CRWs) are handled by a supervisor part (macro **MCRAS**) and a set of RAS transients (phases **\$\$RASTxx**) which are loaded and executed in the RAS transient area. The RAS transient area is located within the supervisor behind macro MCRAS and preceded by eye-catcher **R-TRANSIENT AREA**.

### Machine Check Analysis and Recording

Machine Check Analysis and Recording (MCAR) responds to Machine Check Interrupts (MCIs), attempts recovery and provides operator messages on SYSLOG. Machine check error information is written to the recorder file IJSYSRC by the RAS transients.

In general, a machine check is caused by a machine malfunction and not by data or instruction. Hardware provided mechanisms first try to recover from this malfunction. If these are successful, a machine-check-interruption may or may not be raised indicating the recovery condition. If these are not successful, either a check-stop state is entered (when an uncorrectable malfunction occurs and the machine is unable to recognize a specific machine-check-interruption condition) or a machine check interruption is raised.

#### Hardware actions

A MCI causes following actions to be taken by the hardware:

The PSW reflecting the point of interruption is stored as the MC old PSW into low core storage together with some other valuable information (see Figure 151). Processing continues with the MC New PSW fetched from real location 480 (x'1E0'). This MC New PSW is a z/Architecture PSW and points to the z/Architecture Entry Point of the VSE Machine Check Handler.

Note: The S/390 MC New PSW at real location 112 (x'70') is still maintained (although not used) and also points into the VSE Machine Check Handler but behind its z/Architecture prolog.

Figure 151. MC interruption provided information (z/Architecture mode)

Information provided	Start location	Length
MC Interruption code	232 (x' E8')	8
External-damage code	244 (x' F4')	4
Failing-storage address	248 (x' F8')	8
MC old PSW	352 (x' 160')	8
Floating Point Registers	4608 (x'1200')	128
General Registers 0 - 15	4736 (x'1280')	128
Fixed-logout area	4864 (x'1300')	16
CPU Timer	4904 (x'1328')	8
Clock Comparator	4912 (x'1330')	8
Access Registers 0 - 15	4928 (x'1340')	64
Control Registers 0 - 15	4992 (x'1380')	128

For Hardware-related details on *Machine Check Handling* please refer to *IBM z/Architecture Principles of Operation*.

### Software (VSE) actions

The VSE MC new PSW points to supervisor label MACHEKZ, the entry point of the resident MC Handler (in macro MCRAS).

First of all, the "old" S/390 MC OLD PSW at location x'30' is emulated from the current z/Architecture MC OLD PSW. This is done in order to avoid that system and vendor programs have to change their programs.

Then, the MC is enqueued into an MC queue. This queue holds up to 5 entries. Up to a maximum of five successive machine checks may be queued at a time. Each MC entry is filled with all HW provided information together with other information (like current task, jobname, TOD, etc.) that is needed later on by the RAS system task. This MC queue entry is held in-use until the RAS system task has finished its processing.

Then, the HW provided MC information is analyzed. Dependent on the severity of the MC, one of the following actions are taken:

- terminate system (hardwait)
- terminate task (cancel)
- continue processing (with or without information msg).

Following is a (not-complete) list of conditions which lead to these actions:

1. Conditions for system termination:

- no MC interruption code stored at location x'E8'
- system damage (location x'E8' shows x'80')
- instruction processing damage which is not Backed Up (location 'E8' shows x'40' but location x'E9' does NOT show x'02')
- channel subsystem damage (location x'E8' shows x'08')
- warning condition (location x'E9' shows x'10')
- uncorrected storage or storage key errors (location 'EA' shows x'80' or x'20') with invalid failing-storage address
- uncorrected storage or storage key errors (location 'EA' shows x'80' or x'20') with failing-storage address within Supervisor/SVA routines
- PSW and Register Validity bits are OFF in MCIC (location x'EA'-x'EB') when PSW is in Supervisor State
- more than 10 MCs within 10 minutes (not including MCs with CRW-pending-indication).

*Action:* Post C'A' in location 0 (system termination code) and the emergency exit bit (X'08') is posted in the RAS Linkage Area (see Figure 230 on page 508)

## 2. Conditions for task termination:

The system can continue but the damaged task is to be canceled.

- uncorrected storage or storage key errors (location 'EA' shows x'80' or x'20') with failing-storage address in private partition
- PSW and Register Validity bits are OFF in MCIC (location x'EA'-x'EB') when PSW is in Problem State

*Action:* Activate RAS system task and branch to the cancel routine to cancel the task.

## 3. Soft machine check (if none of the above conditions is present).

Only recording is required for errors from which hardware recovered successfully.

*Action:* Activate RAS system task (if not already active) and return to the interrupted code by loading the machine check old PSW.

Nonresident machine check handling is described in *z/VSE Supervisor Diagnosis Reference Error Recovery and Recording Transients*, SC33-6326.

## Channel Report Word (CRW) Handler

When a Machine Check is raised with 'CRW Pending' indication (location x'E9' shows x'40'), the resident MC handler sets a 'CRW-to-Process' Flag into the RAS linkage area (RASLINK, Figure 230 on page 508).

During the next allbound cycle, a dispatcher appendage in MCRAS (label MCCRWALB) gets control. If the RASLING 'CRW-to-Proces' Flag is ON, all pending CRWs are retrieved from the Hardware (using the STRCW instruction) and stored into the CRW queue.

This CRW queue has 5 entries, where 1 entry is used for one allbound cycle. Each entry may hold up to 32 CRWs.

After having retrieved all pending CRWs, the resident CRW handler in MCRAS performs special actions for the various CRW reporting sources (RSC) and error-recovery codes (ERC), with one exception: When the Channel Subsystem reports

'Event Information Pending', the RAS transients perform the necessary actions to retrieve the event information from the channel subsystem. For an overview on reporting sources, error-recovery codes and VSE actions see Figure 152.

Later on, the RAS system task is posted to record the CRWs into the System Recorder File and to issue a console msg, if required.

Figure 152. CRW handling (Overview)

RSC	ERC	Handling	Message
Subchannel	Available	-	-
Subchannel	Installed Parameters Initialized	Update path masks and CHPID string in PUBX entry of affected subchannel. Set interrupt subclass 3 and enable sub-channel. Call sense task path verification.	-
Subchannel	Installed Parameters Modified	Update path masks and CHPID string in PUBX entry of affected subchannel. Call sense task path verification.	-
Subchannel	Available	-	-
Channel Path	Terminal	Clear subchannel (CSCH) for all devices using the affected channel path and have a function pending. Reset path masks in all PUBX entries that are using the affected channel path. Reset the affected channel path (RCHP).	0T34
Channel Path	Permanent error with facility initialized	Update path masks and CHPID string in all PUBX entries that are using the affected channel path.	0T33
Channel Path	Permanent error with facility not initialized	Update path masks and CHPID string in all PUBX entries that are using the affected channel path.	0T33
Channel Path	Initialized	Update path masks and CHPID string in all PUBX entries that are using the affected channel path. Call sense task path verification.	-
Channel Path	Temporary error	-	0T31
Channel Subsystem	Event Information Pending	(in RAS transients) Retrieve the event information from the hardware and record it into the recorder file.	-
Configuration Alert Facility	Temporary Error	-	0T32
Monitoring Facility	Temporary Error	-	-

## Channel Check Handler (CCH)

The resident CCH gains control from the I/O interrupt handler and (with the exception of 3590 TPA devices) it handles only interface control checks or channel control checks. Channel chaining and channel data checks are handled by the I/O supervisor.

The channel subsystem supplies additional channel check information in the interruption-response block (IRB). The extended-status word 0 (IRBESW0) from the IRB is copied into the ECSW location.

The ECSW is inspected to determine if enough information is valid to isolate the damage to either a channel or a device or if a system termination condition exists. For each channel check an error entry in the PUB extension is used to save error and recording information. If channel and device information is valid the error entry of the corresponding PUB is used.

If a channel damage or a propagated external damage condition exist, the error entry of the first busy device not queued in error on the indicated channel is used for RAS processing. Any other busy device will be restarted if it is a DASD device or set queued in error for non-DASD devices.

For channel checks on disk devices the recovery actions are initiated by the resident CCH. After recovery is done, the error entry is completed and chained to the RAS error chain. The RAS task is posted and control is given to the dispatcher. For an unsuccessful recovery the task in error is canceled.

For channel checks on non-disk devices the error entry is completed, enqueued to the RAS error chain, the RAS task is posted and control is given to the dispatcher. Device-dependent recovery actions and recovery-dependent cancel actions are performed by the RAS monitor and the R-transients.

For 3590 TPA devices, in addition to the RAS error entry a CC error queue entry is allocated and filled with all information needed later on by the RAS system task to recover/record CC errors for these devices.

When a system termination condition is detected, the emergency exit bit is posted in the RAS linkage area (see Figure 230 on page 508) and the RAS task is entered. The applicable termination code is posted at storage location 0. The following list gives the termination codes for the various types of disastrous channel errors:

- B Irrecoverable channel check on fetch.
- C Irrecoverable channel check on paging channel.
- E ECSW not stored.
- G Channel address invalid.
- H Channel check on log with RASMSG.

Nonresident channel check handling is described in *z/VSE Supervisor Diagnosis Reference Error Recovery and Recording Transients*, SC33-6326.

	Record error	Message on SYSLOG	Terminate System	CLRCH	HIO CLRIO	Recovery action
No ECSW stored		X	X			
Interface inoperative	X	X		X		X
CUA valid	X	X				X
RECOVERY ACTION VERIFICATION						
	Retry channel program	Post error in CCB	Cancel channel user			
User own error recovery		X				
Channel program retryable	succ.					
	unsucc.	X	X			
User accepts I/O error	succ.					
	unsucc.	X				

Figure 153. Channel Check Handling Overview

## Recovery Transients and RAS Monitor

The recovery transients (R-transients) perform machine check and channel check recovery and recording.

The RAS monitor is a supervisor resident control program which

- Dequeues error blocks from the RAS error chain.
- Moves error information to the work ERPIB.
- Fetches R-transients into the RTA.
- Schedules I/O requests from the RTA.
- Performs services for the R-transients.
- Provides an exit interface from R-transients.

The RAS monitor table (RASTAB, Figure 231 on page 509), the RAS linkage area (RASLINK, Figure 230 on page 508) and the Error Recovery Procedure Information Block (ERPIB, Figure 232 on page 511) contain the necessary information for the RAS monitor and the R-transients.



## Queues for Machine Checks, Channel Checks and CRWs

Every Machine Check, every Channel Report Word, and every Channel Check for a 3590 TPA device, is stored in a corresponding queue by the resident MC, CC and CRW handlers in MCRAS.

The MC and the CC queue may hold 5 entries, the CRW queue may hold 5 times 32 CRW entries in parallel. Queue-full conditions for the CC and the CRW queues reside in information-lost conditions (with a corresponding console msg), whereas a queue-full condition for the MC queue results in a hardwait.

### Queue control

Figure 154. MCRAS queue control

Offset	Supervisor Label	Description
0 (x00)	MCSTKNUM	Number of MC/CRW/CC Queue entries
2 (x02)		reserved
4 (x04)	MCSTKQUE	MC queue: Ptr to 1st entry in use
8 (x08)	MCSTKBEG	MC queue: Ptr to begin of queue
12 (x0C)	MCSTKEND	MC queue: Ptr to end of queue + 1
16 (x10)	MCSTKNXT	MC queue: Ptr to next free queue entry
20 (x14)	MCRWSQUE	CRW queue: Ptr to 1st entry in use
24 (x18)	MCRWSBEG	CRW queue: Ptr to begin of queue
28 (x1C)	MCRWSEND	CRW queue: Ptr to end of queue + 1
32 (x20)	MCRWSNXT	CRW queue: Ptr to next free queue entry
36 (x24)	MCRWSLST	CRW queue: Ptr to last CRW in current Q-entry
40 (x28)	CCSTKQUE	CC entry: Ptr to 1st entry in use
44 (x2C)	CCSTKBEG	CC entry: Ptr to begin of queue
48 (x30)	CCSTKEND	CC entry: Ptr to end of queue + 1
52 (x34)	CCSTKNXT	CC entry: Ptr to next free queue entry

How to locate these queue addresses?

1. LOCATE 'RAS-SCHIB
2. Subtract x'38' from the given address
3. New address points to begin of above structure (MCSTKNUM)

The queue entries in-use are forward-chained with a forward pointer at offset 0.

A queue entry is held in-use until all required processing has been done by the resident MC/CRW/CC handler and the RAS system task.

For a layout of a queue entry please refer to

- Macro MAPMCSTK (for a MC queue entry)
- Macro MAPCRWST (for a CRW queue entry)
- Macro MAPCCSTK (for a CC queue entry)



---

## Job Accounting

The support for job accounting is always generated in the VSE/AF Supervisor and is optionally activated at IPL time by SYS JA=YES.

Job accounting is associated with the following data areas:

- Some fields in the system communication region SYSCOM
- The job accounting common table ACCTCOMN
- Some fields in the partition communication region COMREG and in the Partition Control Block (PCB).
- For each partition the job accounting partition table ACCTABLE
- A set of device usage and SIO counters associated with the PUB-extension PUBX and the PCB.
- A 1K-user save area.

All data areas are allocated below the 16MB line (RMODE 24).

Job accounting logic consists of three distinct parts:

- The allocation and initialization of accounting areas and fields at IPL time and during dynamic partition allocation.
- The maintenance of accounting information at system run time.
- The interface to the user written accounting routine \$JOBACCT.

## Initialization

The system and static partition related tables are allocated and initialized at IPL time, the ones for dynamic partitions during dynamic partition allocation. During IPL most work is done by phase \$INITSYS, which is executed after all system options have been specified. When \$INITSYS is invoked, the following initialization relevant to job accounting is already done:

- SYSCOM.IJBFLG02.IBJA is set on if JA=YES was specified.
- A PUBX is allocated and initialized for every added device.
- The total number of added device is stored in SYSCOM.IJBNDEV.
- The total number of added 'partition sharable' devices is stored in SYSCOM.IJBNSDEV.

DASD devices, unit record devices and the SYSLOG device are considered as partition sharable. Unit record devices are included because they can be used as dummy devices for VSE/POWER in more than one partition.

If SYSCOM.IJBFLG02.IBJA is on,

- \$INITSYS
  - Allocates a 1K-user save area and saves its address in ACCTCOMN.ACCTUSEP.
  - Calculates the length of ACCTABLE depending on SYSCOM.IJBNDEV and saves it in ACCTCOMN.ACCTABLN.
- \$INITSYS or dynamic partition allocation (\$IJBSSM)
  - Allocates and initializes one ACCTABLE per partition in pageable system GETVIS space.
  - Saves the address of each ACCTABLE in COMREG.JAPART.
  - Sets COMREG.JCSW1.JASWITCH off as an external indication (mainly for job control and VSE/POWER) that job accounting is active.

- Sets PCB.PCBJAPTR = A(PCB) as an internal indicator that job accounting is active.
- Allocates in fixed system GETVIS space strings of usage and SIO counters for partition sharable devices, one string per partition, saves the address of each string in PCB.PCBCNT and the offset within the string of the SIO counter for each device in PUBX.PBXJAOFF.

## Maintenance

At system run time, CPU time and SIO counters are maintained in internal fields, which are not directly accessible to the user.

For CPU time accounting, short time intervals (typically between dispatching and interrupt times) are measured with the CPU Timer in units of 16 microseconds and assigned to a partition, whenever possible, or to the system as overhead times. Time intervals with the CPU in wait state are accumulated in a separate allbound time counter.

The criterion for assigning a time interval to a partition is that the time interval represents a reproducible portion of productive work for that partition. System activities, which do not fall under this categories, are the following:

- Paging
- Channel scheduling
- Hardware error recovery
- First level timer interrupt processing
- Attention routine processing (operator commands)

CPU time intervals assigned to a partition are accumulated in the field PCB.RUNTIME. The corresponding field in the system PCB is used to accumulate overhead time intervals. System wait state intervals are accumulated in the low core field SBNDTIME.

Field PCB.PCBJAPTR points to the PCB to which the current time interval is to be assigned. For partition PCBs, PCBJAPTR may point to the PCB itself (partition time) or the system PCB (overhead time). For the system PCB, PCBJAPTR may point to the PCB of the service owner (partition time) or to the system PCB itself (overhead time). System tasks, whose processing is always counted as overhead time, are flagged by TIB.TIBFLAG2.OVHIND.

Whenever a CPU Timer interrupt occurs or a GETJA request is issued (see below), the contents of the fields PCB.RUNTIME and SBNDTIME are transferred to another set of internal fields in each partition PCB, namely PCPUTIME, POVHTIME and PBNDDTIME. PCB.RUNTIME in the partition PCB is simply added to PCPUTIME. The accumulated overhead PCB.RUNTIME in the system PCB is distributed among the fields PCB.POVHTIME of all active partitions in proportion to their PCB.RUNTIME values. SBNDTIME is distributed in equal parts among the fields PCB.PBNDDTIME of all active partitions.

SIO counters are maintained for all devices in internal fields associated with each PUBX. For devices, which are not partition sharable, a single counter PUBX.PBXJACNT is sufficient. For partition sharable devices, there is one internal SIO counter per device and partition located at (PCBCNT)+(PUBX.PBXJAOFF).

The SIO counter is updated immediately after a successful SIO and, for spooled dummy devices, after successful invocation of the VSE/POWER SVC 0 appendage. SIOs for system tasks with TIBFLAG2.OVHIND on as well as those associated with the logical unit SYSUSE are not counted.

## User Interface

Whenever a job step is completed, job control invokes the user accounting routine \$JOBACCT. Accounting data is passed to the user in the accounting partition table ACCTABLE. The transfer of the internal counters into the ACCTABLE is controlled by the macro GETJA, which is invoked by job control at well defined points within job processing, in order to restrict the data in ACCTABLE to single job steps. For details on the function of GETJA refer to the internal macro descriptions in Appendix B.

The GETJA routine is also internally invoked by SVC 112 (X'70' - MSAT macro), to save the SIO counter of a device, which is not partition sharable, into ACCTABLE whenever device ownership is released.



---

## Software Re-IPL

The main purpose of software re-IPL is to automatically re-IPL the system during an automatic installation of a z/VSE system. It is activated via SVC 31 at the end of stand-alone processing after the SYSRES has been restored.

Software re-IPL may also be requested by the operator REIPL command.

```
REIPL cuu,LOADP=.....
```

**Note:** If the device to be REIPLed is a SCSI device, specify the cuu of the IPLed **FBA-SCSI disk**, although the load device actually is the attaching FCP adapter.

## Execution

It depends on the architecture of the disk to be IPLed and also on the architecture of the processor how software IPL is done. The following environments and hardware features have to be considered:

- LPAR or basic mode
- VM
- CCW based IPL as for ECKD or FBA SYSRES
- LD IPL as for SCSI SYSRES
- PD IPL

Traditional CCW based IPL is handled in SGUNATT, LD IPL and PD IPL for SCSI devices in SGREIPL.

The Re-IPL routine in SGUNATT performs the following functions:

- Simulates the Hardware IPL.
  - Running in an LPAR
    - Clears the TLB (translation look-aside buffer) and initializes the control registers.
    - Resets the I/O subsystem.
    - Reads the bootstrap record from disk and passes control to the bootstrap phase by loading the IPL PSW from location zero.
  - Running under VM
    - Passes the CP command 'IPL cuu' to VM with the VM Diagnose instruction.

The Re-IPL routine in SGREIPL performs the following functions:

- Running in an LPAR supporting PD IPL:
  - Builds the IPL Parameter List according to the architecture.
  - Issues DIAGNOSE X'308', function 5 to pass the IPL Parameter List which is needed for the subsequent LD IPL to the processor.
  - Issues DIAGNOSE X'308', function 3 to request PD IPL.
- Running in an LPAR without PD IPL support:

- Enters a wait state, because software re-IPL cannot be performed. IPL is required from the HMC.
- Running under VM
  - Issues DIAGNOSE X'08' instruction to pass the CP command  
SET LOADDEV PORT wwpn LUN lun SCP NEW '...'  
and tell VM the connection path to the SCSI disk to be IPLed.
  - Issues DIAGNOSE X'08' instruction to pass the CP command  
IPL fcp\_cuu  
to have VM start LD IPL.

## Invocation

### SVC 31

In the SVC 31 processing, an authorization check is done. If the requestor has no authorization, the caller is canceled with illegal SVC. Control is passed then to either SGUNATT or SGREIPL.

Input Parameters:

- Call during automatic installation:
  - Configuration data - such as cuu of SYSRES, SYSWK1 and operator console, connection paths if applicable, load parameters - are transferred from the stand-alone session to the re-initialization
    - in main storage at locations that remain unmodified (ECKD, FBA),
    - in the REIPL record of the disk being IPLed (SCSI).

The data are prepared and put in place by the caller before issuing SVC 31.

- Call from REIPL command processing:
  - None.
  - The re-IPL data are set up by the SVC routine
    - in a fixed storage location (ECKD, FBA),
    - in the SCP data part of the IPL Parameter List on a processor that supports PD IPL (SCSI).

Software Re-IPL uses the following data areas:

- Hardware locations as defined in SGLOWC,
- Re-IPL control block UNATTCB,
- IPL Parameter List MAPIPLPL (Program Directed IPL),
- SCSI re-IPL control block MAPREIPL.







---

# Console Support

This chapter contains only information that is intended for public access and is not covered elsewhere in this or in other externally available VSE publications.

## Overview

This section describes the data flow, structure and overall processing related to the console support. The data flow is shown in Figure 155 on page 384.

**IPL Processing:** IPL supports the Integrated Console as communication device, in addition to 3215 and 3270 type devices. Whenever Console Integration is available, it allocates a PUB for a device of type CONS with a dummy device address, to which SYSLOG can be assigned (during IPL or later).

The selection of the IPL communication device depends on ASI definitions, device availability and the IPL load parameter. The access to the selected device during early IPL processing, before the supervisor is loaded, is via synchronous interfaces (stage I). After the supervisor is loaded, the Console Router and the console modules, that support the selected device (Integrated Console or CRT/Line Mode) and redisplay from Console Router storage, are loaded in real storage and initialized (stage II), and communication switches to the standard interfaces (SVC 0/15, WTO/WTOR). After the SVA is loaded, console processing is taken over by console modules loaded in the 31-bit SVA (stage III) and the real area allocated during stage II is reused.

IPL messages are collected in the Console Router queue, for later logging as soon as the HC File is opened.

**Console Router:** The Console Router provides the following functions:

- MVS compatible WTO/WTOR support
- WTO/WTOR (VSE/ESA V1 level)
- Application control of message deletion (DOM macro)
- Analysis of console I/O channel programs (SVC 0/15)
- MVS compatible console interfaces (MCSOPER, MCSOPMSG and MGCRE macros)
- SVC 30 (command input)
- Console buffering asynchronous operator communication
- Routing to multiple consoles
- Interface to CMS users via VMCF
- Console recovery
- Command interface to Attention Routine
- Queuing of AR commands
- Logging to HC File
- Support for HC File redisplay commands
- Vendor exits

The Console Router also calls unique OCCF functions, when active:

- Message suppression and automated reply
- Message/reply translation

The Console Router becomes active during IPL, immediately after the supervisor is loaded. It maintains a set of queues for all console traffic in transit, including:

- Messages waiting for delivery to one or more consoles
- Action messages, held until deleted by the operator or by the originating application

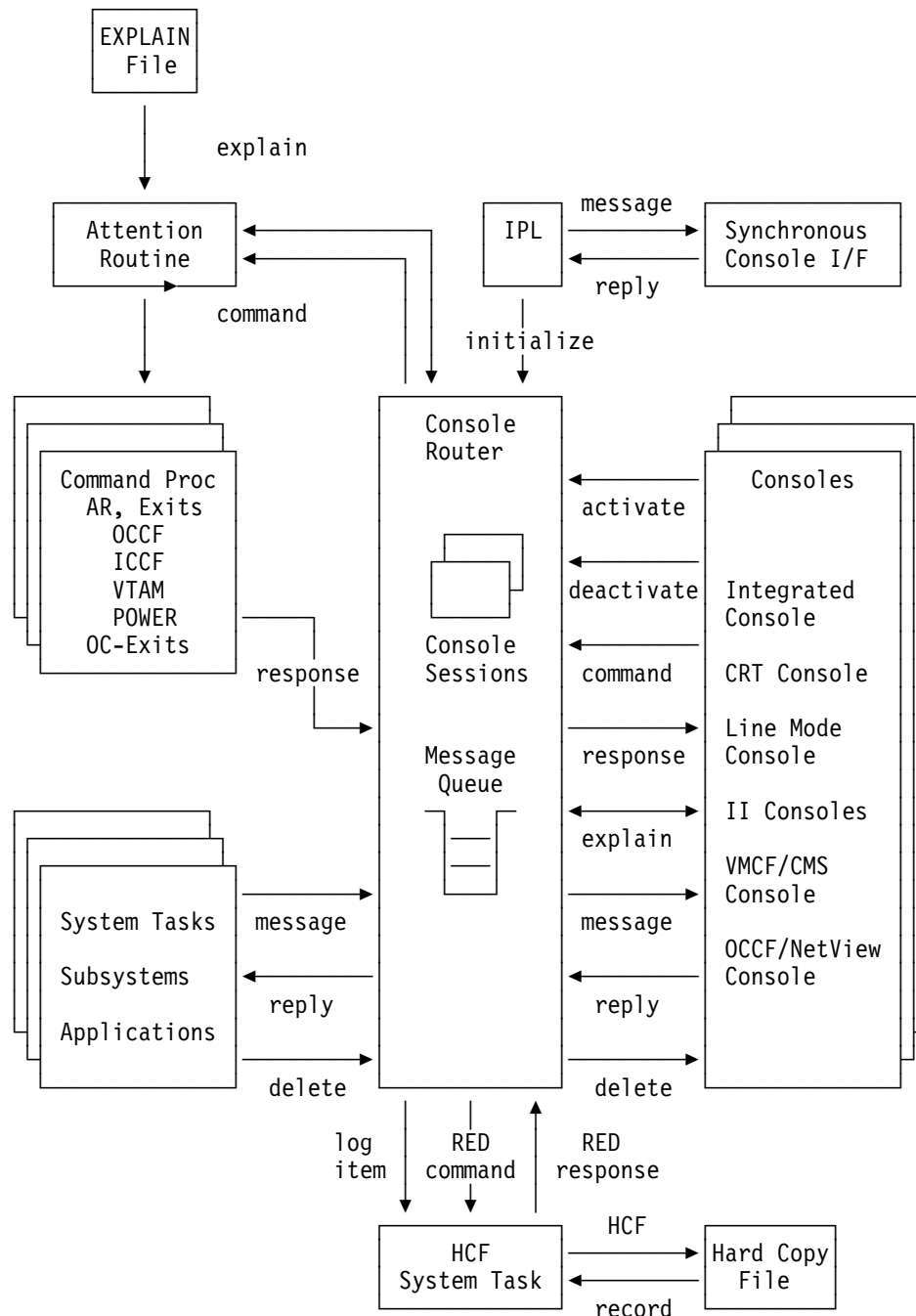


Figure 155. Console Data Flow

- Decision messages awaiting a reply
- HC File records to be logged
- Commands to be processed by the Attention Routine or by the HCF task.

Queue storage is allocated in 31-bit system GETVIS space.

The Console Router supports a set of interfaces for the data flows shown in Figure 155. Most of these interfaces are implemented by calls to Console Router services (SVCs, Program Calls and branch entries). Since the Console Router has no task of its own, all control flows start from an external task calling the Console Router and are executed under that task.

The application 'message' and 'reply' flows (left side of Figure 155) are based on the SVC 0/15 and the WTO/WTOR interfaces. The 'delete' flow corresponds to the DOM interface.

The console flows (right side of Figure 155 on page 384) are based on MVS-compatible extended-MCS interfaces.

The Attention Routine interface ('command' flow) is called by the AR task to get the next command from the Console Router command queue, and includes the name and ID of the originating console and a command/response correlation token.

The 'explain' flow allows console with explanation support to obtain a pointer to help and message explanation text, that is loaded in virtual storage by an AR service.

The 'log item' and 'RED command/response' flows describe the communication with the HCF system task, that retrieves entries to be logged or redisplayed, as well as redisplay commands, from the Router queue, and enqueues redisplay responses to be routed back to the requesting console.

**Integrated Console:** The Integrated Console is always activated, when the SPDT interface for Console Integration is available, and is used as system console, when SYSLOG is assigned to it, or, in connection with a CRT/Line Mode system console or other master consoles, as alternate console for recovery situations. The support consists of a synchronous component, used during IPL stage I, and a full-function component that runs under the existing SPT system task. The full-function component communicates with the Console Router via the standard MCS interfaces, and exchanges console traffic with the Service Processor using the architected Message Data Block format.

The Integrated Console must be protected by physical security.

**CRT/Line Mode Console:** The CRT/Line Mode Console is activated only when SYSLOG is assigned to a 3270 or 3215 device, and is used in this case as the standard system console. During IPL stage I it is accessed via existing IPL interfaces. As soon as IPL stage II is entered, the full-function code is loaded and executed under the CST system task. Communication with the Console Router is via the standard MCS interfaces. The transition between IPL stages II and III is handled transparently for the user (continuous screen contents).

The CRT console has the same appearance and usability characteristics as the II console. This is ensured by isolating a major portion of the code, that is not sensitive to the environment or I/O characteristics of the console, and by sharing this code with the II console.

The CRT console can be disconnected, in which case the system console role is automatically transferred to the Integrated Console, when available.

The CRT/Line Mode console must also be protected by physical security (no operator logon).

**II Consoles:** The II supports CICS-based master and user consoles, that are independent of the CRT console, but with identical appearance and usability characteristics (ensured by common code). These consoles also communicate with the Console Router via the standard MCS interfaces.

Redisplay is implemented via redisplay commands routed to the HCF task, rather than through synchronous HC File I/O (not tolerable under CICS).

Message explanation text is loaded asynchronously from the Online Message Explanation file via the AR EXPLAIN service.

The access to the master console capability is controlled through II security facilities.

**VM/VSE Interface Routines:** The routing of console traffic between VSE and CMS users in a VM/VSE environment is accomplished via a single VMCF Console. This console functions as a VSE master console, when enabled via a SYSECHO command, and supports a variable number of CMS users, that receive unsolicited messages only through the ECHO or ECHOU option.

All routing to CMS (including redisplay responses) is via MSGNOH/MSG commands submitted to CP by a synchronous Router exit routine. This routine is invoked for all master console messages, and for other messages with explicit destinations not identified by other active consoles. The MCSOPMSG macro is not used for this console.

Input from CMS is supported through the VSECMD interface, that communicates via VMCF with a VSE module running under the SPT system task.

When VSE security is active, access to master console functions by CMS users, other than the SYSECHO userid, is protected by passing the CMS userid and the console authorization of an identical VSE userid to the Router (UTOKEN parameter of MGCRE).

**OCCF And NetView Console:** OCCF message translation and automation functions do not require a partition. They are activated via a system command, passed to OCCF through the AR sub-system interface, and executed by OCCF exit routines invoked by the Router.

The support for the NetView console is available in connection with OCCF and runs in a private partition (static or dynamic), along with the Unattended Node support. It communicates with the Console Router via a unique console using standard MCS interfaces, and with NetView over an XPCC interface. Console traffic is routed to this console based on a routing option set by the OCCF message exit.

The access to the master console capability of the NetView console is controlled through NetView security facilities.

**HCF Task:** The HC File is opened by Job Control, when the first // JOB statement is encountered, and independently of SYSLOG device type. All HC File I/O requests, both for logging console traffic and for re-display purposes, are then handled by the HCF system task. The HCF task is started as soon as IPL stage II is entered, and processes logging requests off the Router queue without interfering with console traffic, except for a synchronization mechanism at end of job step, that guarantees a consistent HC File contents for LISTLOG. Responses to re-display commands are passed through the Router queue and delivered, like other messages, in the standard MDB format to the requesting console.

**Attention Routine:** AR retrieves commands from the Router queue via service routines, that queue commands entered from different consoles. The 'busy' condition presented by subsystems is handled transparently by a synchronization mechanism, with no impact on the processing of other commands.

Access to the Message Explanation File, in response to an EXPLAIN request, is supported by AR via standard VSAM interfaces, that load message explanation text from a specified record into virtual storage. Only the address and length of this area is returned to the requesting console.

## Supervisor Interfaces

All interfaces to the Console Router are accessed through the supervisor (macro SGDOC) via various call mechanisms. Independently of the mechanism the calling task is put into an SVC-like status, before invoking the Console Router entry point in 31-bit SVA with a unique function code in register 0. Control always returns to the supervisor, behind the point of invocation, and the original task status is restored through an appropriate exit mechanism. This is summarized in Figure 156.

Figure 156. Console Router Interfaces

Interface	Call Mechanism	Function Code	Exit Mechanism
New WTO/WTOR	SVC 131, MVS SVC 35	1	DISPMVS
Old WTO/WTOR	SVC 32	2	EXIT
DOM	SVC 131, MVS SVC 87	3	DISPMVS
SYSLOG I/O	BAL from SVC 0/15	4	EXIT
MGCRE	SVC 131, MVS SVC 34	5	DISPMVS
SVC 30	SVC 30	6	EXIT
MCSOPER	PC 1	7	DISPSERV function code 12
MCSOPMSG	PC 6	8	DISPSERV function code 12
Message Clean-up	BAL from SVC 3	9	Return to IOS
End of Task Clean-up	BAL from Terminator	10	Return to Terminator
HCF Initialization	BASSM from \$IJBPHCF	11	DISPSERV function code 12
HCF Get Request	BASSM from \$IJBPHCF	12	DISPSERV function code 12
AR Initialization	BASSM from \$IJBAR	13	DISPSERV function code 12
AR Get Command	BASSM from \$IJBAR	14	DISPSERV function code 12

The Console Router executes as a reentrant supervisor routine (RID = REENTRID). Reentrancy is ensured by allocating 8K of automatic storage for save and work areas for each call.

Two automatic storage areas are pre-allocated during Console Router initialization:

- The first area is fixed and reserved for system tasks that issue messages but do not tolerate page faults (currently applies to the SNS and RAS tasks). Contention on this area is excluded, since Console Router processing for these tasks is guaranteed to be non-interruptible.
- The second area is pageable and used for all other tasks, when available.

When the pre-allocated area is not available, one additional area is allocated dynamically in 31-bit system GETVIS space, and freed when processing completes. When two pageable areas are already in use, the calling task waits until one of them becomes free.

The structure of the automatic storage is shown in Figure 157 on page 388.

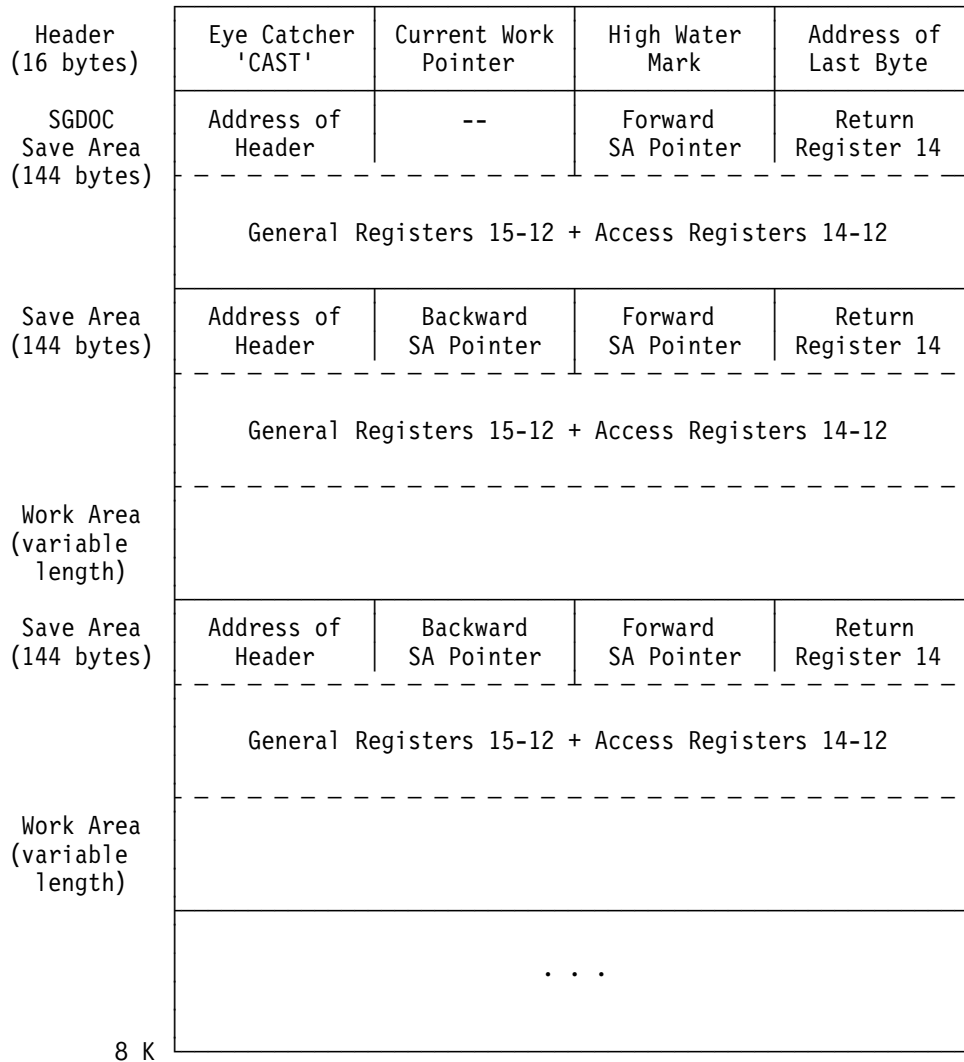


Figure 157. Console Router Automatic Storage

The Console Router returns to the supervisor with return and reason codes in registers 15 and 0, that are interpreted and processed as shown in Figure 158.



Figure 158. Console Router Return and Reason Codes

Return Code	Reason Code	Processing
0	-	Control is returned to caller via exit mechanism. Output registers have been stored by the Console Router in the callers save area.
4	4	The caller must wait for console buffers. The task enters wait state via RESVCX or RWAIT.
4	8	The caller must wait for a reply ID. The task enters wait state via RESVCX or RWAIT.
8	4	The caller has supplied an invalid address. The task is cancelled with cancel code x'25' (exit to ERR25)
8	8	The caller has supplied an invalid SYSLOG channel program. The task is cancelled with cancel code x'39' (exit to ERRGO)
8	12	The caller has supplied invalid parameters. The task is cancelled with cancel code x'21' (exit to ERR21)
8	12	The caller tried to send a message to an unknown destination. The task is cancelled with cancel code x'1A' (exit to ERRGO).
16	-	The mode of operation is REMOTE and a message requiring a reply could not be routed to any destination other than the System Console. A software re-IPL is initiated, since input from the System Console is inhibited in this mode of operation.

The calling task may have to wait for Console Router resources, that are summarized in Figure 159

Figure 159. Console Router Bound Conditions

Bound State	Meaning
DOCABND (x'48')	Wait for automatic storage. The pre-allocated automatic storage was not available, and the limit for dynamically allocated automatic storage has been reached or GETVIS failed.
DOCBBND (x'49')	Wait for Console Router buffers or for a reply ID. This bound condition is used with a wait argument of 0, when the limit of buffers per task was reached, or the overall buffer space could not be expanded or GETVIS failed. The address of the task's bit map of outstanding replies (TCSRBM in TCB-extension) is used as a wait argument in the latter case.
DOCQBND (x'67')	Wait for Console Router page-fault gate. The task executing a gated Console Router sequence was interrupted by a page fault, and the current task must wait until the gate owner resumes and exits from the gated sequence.

## SYSLOG I/O Interfaces

**Rules For SYSLOG Channel Programs:** Operator messages may be issued by means of an I/O request (SVC 0/15) for the logical unit SYSLOG, with a channel program suitable for a printer keyboard device. This interface is maintained for compatibility reasons. Conversion to WTO/WTOR is recommended.

The rules for a valid SYSLOG channel programs are summarized in the table below.

Figure 160. Supported Channel Commands and CCW flags

Channel commands		CCW flags	
<b>01</b>	Write, no carrier return	<b>CD</b>	Chain data of next CCW
<b>09</b>	Write, auto carrier return	<b>CC</b>	Chain command code of next CCW
<b>0A</b>	Read inquiry	<b>SLI</b>	Suppress incorrect-length indication
<b>0B</b>	Audible alarm		
<b>03</b>	No operation		
<b>04</b>	Sense		
<b>08</b>	Transfer in channel		

Other channel commands and CCW flag settings will be rejected.

The following table shows the actions which are performed then and list also other conditions that lead to a rejection of a SYSLOG channel program.

Figure 161. Rejection of a SYSLOG channel program

Condition	Action (refer to action list below)
Unsupported channel command used	Act. 3 (unit check)
Read-CCW (CommandCode x'0A') with command chaining or data chaining	Act. 3 (unit check)
Unsupported CCW flag used	Act. 1 (channel program check)
EXCP REAL for non-system tasks (V=R addresses assumed for system tasks)	Act. 1 (channel program check)
CCW not on Doubleword-boundary	Act. 1 (channel program check)
A TIC is directly followed by a TIC	Act. 1 (channel program check)
CCW address or CCW data buffer invalid	Act. 2 (channel protection check)
Data length in CCW is 0	Act. 1 (channel program check)
<b>Read CCW (x'0A') or Sense CCW (x'04')</b> from non-system task with <ul style="list-style-type: none"> <li>• data area not within partition limits</li> <li>• if task owns LTA and data area not in LTA</li> <li>• if VTAM CICS POWER task and data area not in SVA</li> </ul> (Checking done in sequence given!)	Act. 2 (channel protection check)
More than 32 CCWs in a CCW chain	Act. 1 (channel program check)

**Rejection of a SYSLOG channel program** means:

- the intended message is suppressed
- and one of the following actions is performed:

**Action List:**

1. Simulate Channel Program Check
  - In user CCB/IORB
    - set IORBCNT = 0
    - post I/O complete, unrecoverable I/O error, channel end, device end
    - post program check
  - If unrecoverable I/O errors are not accepted
    - cancel task with cancel code x'39' (Invalid CCW chain for SYSLOG)
    - msg 0V12I (Error in SYSLOG channel program) is issued
2. Simulate Channel Protection Check
  - In user CCB/IORB

- set IORBCNT = 0
  - post I/O complete, unrecoverable I/O error, channel end, device end
  - post protection check
  - If unrecoverable I/O errors are not accepted
    - Cancel task with cancel code x'25' (Invalid address)
3. Simulate Unit Check
- In user CCB/IORB
    - set IORBCNT = 0
    - post I/O complete, unrecoverable I/O error, channel end, device end
    - post unit check
  - If unrecoverable I/O errors are not accepted
    - cancel task with cancel code x'39' (Invalid CCW chain for SYSLOG)
    - msg 0V12I (Error in SYSLOG channel program) is issued

The next table shows some special processing for a SYSLOG channel program.

Figure 162. Special Processing of a SYSLOG Channel Program

Condition	Action (refer to action list below)
Execution of Channel End appendage requested	Ignored
<b>Sense CCWs</b> (Command Code x'04')	returns <b>zero sense data</b> (x'00') (CCW not passed to consoles)
<b>NOOP CCWs</b> (Command Code x'03')	are skipped (and not passed to any console)
Write-CCW with data length of > 70 characters.	A new line is forced after 70 characters (or at a meaningful separation position, resp.). The text of a following write (x'01') CCW will be appended, a following write with carrier return (x'09') will start the next new line.
SYSLOG channel programs with more than 12 lines (either specified via carriage return CCWs x'09' or forced by the preceding rule) or with more than an (accumulated) message length of 700 characters	Message will be truncated (with an additional info message 0D96I <b>over-laying</b> the last message line)

**Note:** Format-1 CCWs in SYSLOG channel programs are supported.

**Assigning Routing and Descriptor Codes:** Routing and descriptor codes for messages issued via the SYSLOG I/O interface are assigned by the system, based on an exception list and on defaults derived from other criteria, as described in Figure 163.

Figure 163. Routing and Descriptor Codes for Message issued via SYSLOG I/O

Message Criteria	Routing Codes	Descriptor Codes
<b>Messages with a message code of the form 'xxxna' (x alphanumeric, n numeric, a = W or A or E or D or I):</b>		
Found in exception list	from list	from list
Action code W	1	1
Action code E	1	3
Action code A or D, by system task/subsystem/LTA	1	2
Action code A or D, by AR task	-	-
Action code A or D, by partition/SVA	11	-
Action code I, for system task/subsystem	2	4
Action code I, for AR task	-	5
Action code I, for partition	11	6
<b>Other messages:</b>		
Reply needed	11	-
No reply needed, by AR task	-	5
No reply needed, not by AR task	11	6

Note: The criteria for messages with action code D refer to the message origin (task, LTA), whereas as those for action code I refer to the part-ID (service owner) in the message prefix (see also "Message Presentation" on page 394).

Whenever a message is also defined in the following exception list, the routing and/or descriptor codes from this list will overwrite the routing and/or descriptor codes derived from the above rules.

Figure 164. SVC 0 Message Exception List

Msg Code	RC	DC
1Q05I	set by above algorithm	11
1Q15I	set by above algorithm	11
1Q2CI	set by above algorithm	11
1Q2DI	set by above algorithm	11
1QB5I	set by above algorithm	11
4n44D (n = 2,...,9)	set by algorithm	2

Messages issued by the AR task via SYSLOG I/O are additionally tagged with the console-ID of the console, where the current command was entered, as if they had been issued via WTO/WTOR with the CONSID parameter. All responses (no READ CCW) to the same command are also connected, like WTOs with CONNECT parameter, and a last line with linetype E is added by the system, when the response is complete.

When a message is issued for a partition with active ECHO option (for the meaning of 'for a partition' see "Message Presentation" on page 394) and routing code 11 was set according to above criteria, routing code 11 is reset and the ECHO userid is inserted for CONSNAME.

When CONSID or CONSNAME were inserted, for one of the reasons mentioned above, and no console with that name is active at the time the message was issued, nor a CMS user with that userid, the

message is processed anyway, if routing codes other than 11 are on, or suppressed if it does not require a reply and is not immediately followed by a stand-alone read issued by same task. If a reply is required, unrecoverable I/O error is posted in the CCB, when requested, or the issuing task is cancelled with cancel code x'1A' (I/O error).

## Message Presentation

The system edits the message text, as supplied via WTO(R) or the SYSLOG I/O interface, and assigns default color, highlighting and intensity attributes, independently of the device used to display the message.

Message editing consists of splitting text lines that do not fit within the standard width of 70 characters (applies only to single line WTO(R) and SYSLOG I/O messages), and of adding a message prefix in front of the first line. The line splitting algorithm is the same as today. The message prefix is built, like today, in the form

**partition ID [+|-] reply ID**

where:

**partition ID** takes as today one of the values AR, BG, Fn, cn (c = dynamic class), and identifies the service owner partition. When the message is issued by a system task or by VSE/POWER, this is the partition that is being serviced, when applicable. In all other cases, it is the partition of the task that issued the message.

**+** indicates a message that needs an immediate reply, because a system resource (e.g. the LTA) is being held.

**-** indicates any other message needing a reply.

**reply ID** takes the form **rnnn**, where **nnn** is a 3-digit numerical value derived from the task ID of the task that issued the message (algorithm see below), and **r** (new) is a digit 0-9 used to distinguish multiple outstanding replies for the same task. The additional digit is needed for tasks that are allowed to have more than 1 outstanding reply. Primarily these are subsystems, like CICS or POWER, that use internal tasking.

Figure 165. Algorithm to calculate "nnn" in reply id

Task	Task Id	"nnn"
Main tasks in static partitions	x'21' - x'2C'	000 - 011 (*)
Selected system tasks	x'01' Sense	013 (*)
	x'0B' ERP	014 (*)
	x'20' AR	015 (*)
Other system tasks	x'02' - x'0A'	016 - 024
	x'0C' - x'1F'	025 - 044
Subtasks or main tasks in dynamic partitions	x'002D' - x'0200'	045 - 512 (**)

(\*) same as in VSE/ESA V.1.

(\*\*) 512 (or x'0200') is the highest task-Id that can be in use only when support for more tasks has been activated with SYSDEF SYSTEM,NTASKS=MAX. If support for more tasks is not active, the highest task-id is 255 (or x'00FF').

The default background color is black, and the default message attributes are determined by the criteria shown in the following table. Both may be modified by OCCF and Message Processing Vendor exits. Presentation attributes of other display fields are controlled by console applications.

Figure 166. Default Message Presentation Attributes

Criteria	Color	Highlighting	Intensity
Descriptor code 1 or 11	red	none	high
Descriptor code 2 or reply needed	white	none	high
All other cases	green	none	normal

## HCF System Task

The HCF task is responsible for logging message traffic on the Hard Copy File and for responding to REDISPLAY commands. It is activated when IPL stage II is entered, although the HC File is not yet open, to enable REDISPLAY from the Console Router queue.

After activation, the HCF task waits in the supervisor on an ECB, that is posted by the Console Router when there is work to do, or when the transition to IPL stage III is to take place. All actual processing is done by the phase \$IJBPHCF, residing initially in IPL space and later on in 31-bit SVA. Only persistent status information is maintained within the supervisor (CRTGEN macro).

On unexpected error conditions, a system task error exit in the supervisor is invoked, causing recovery and retry logic to be driven. When recovery fails to the point that no useful work can be done any more, the HCF task terminates after issuing a final error message.

## CST System Task

The CST system task manages the 3270 and 3215 system console. Depending on the type of IPL communication device, it is activated when IPL stage II is entered, or otherwise after the transition to stage III has occurred.

After activation the CST task calls phase \$IJB CRT, residing initially in IPL space, when needed, and later on in 31-bit SVA, and never returns to the supervisor. Only those data areas that must be below 16M (e.g. CCB and CCW's) are maintained within the supervisor.

When the device managed by the CST task (SYSOCDEV) presents an attention interrupt, IOS calls a CST appendage that in turn posts the CST task for pending console input. All I/O operations are done via the standard SYSIO interface.





---

## Chapter 3: Diagnostic and Debugging Aids



---

## Diagnostic Aids

---

## Fixed Storage Locations in Processor Storage (Low Core)

In z/Architecture mode, the prefix area (low core) comprises two 4K pages and is 8K in length.

In a multi-processing environment each CPU has its own (8K) prefix area. The layout of the prefix area is standard for any IBM System z processor.

Usage of the z/Architecture prefix area in z/VSE:

- The locations of the ESA/390 new and old interrupt PSWs are not used by the hardware and are used by z/VSE to emulate the ESA/390 PSWs.
- Storage locations in the prefix area (8K) which were used by VSE and which are now used by the hardware have been rearranged.
- To avoid synchronization of the prefix areas in a multi- processing environment, variable system control blocks have been moved beyond the prefix area.

---

## Supervisor Patch Area

Supervisor patch areas are provided for use by IBM programming support representatives. They use those areas if there is a need for installing a local fix to (usually a bypass of) a problem.

There is one 28-byte patch area within the supervisor at label IJBPATCH (X'200' in low core). The first four bytes of this area point to a 300-byte patch area in the high address range of virtual storage. The small area within the supervisor allows coding of a limited number of instructions without the need for a base register in operand addresses.

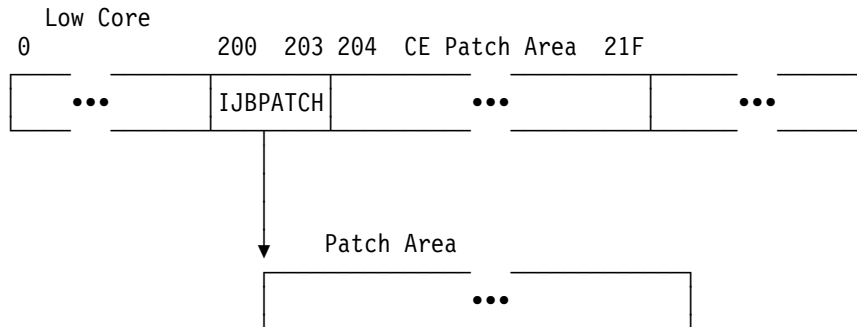


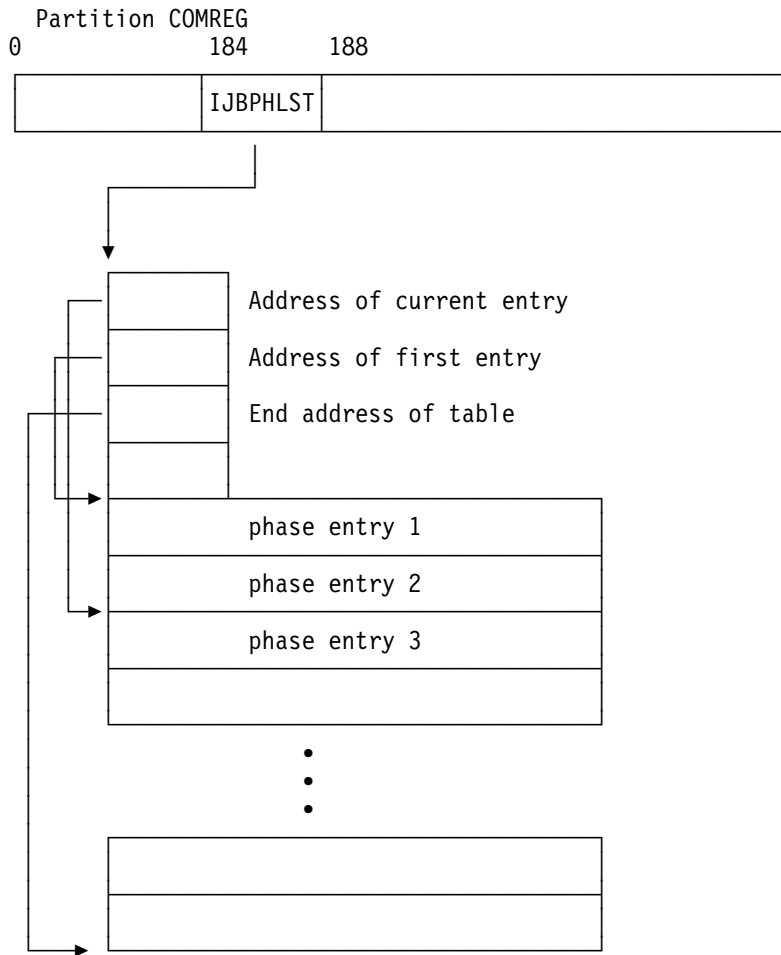
Figure 167. Patch Area Relationship

## Phase Load Trace Table

For every partition an area for the Phase Load Trace Table is allocated in the system GETVIS area. This is done at IPL time for static partitions and during allocation for dynamic partitions. The address of the area can be found at label IJBPHLST in the partition communication region. An area is IJBPHLSL (376) bytes in length.

The supervisor writes a phase load entry into the table each time a phase is loaded into the partition. It is possible to record 15 load requests. If the table is full, the supervisor wraps around and starts from the beginning of the table.

At the end of a job step, the table is cleared by the system.



Layout of a table entry

phase name	load address	phase length	SDT pointer	LDT pointer
0	8	12	16	20
				23

Figure 168. Phase Load Trace Table

## Hard Wait Codes

### Machine Check / Channel Check Wait Codes

Byte 0	Byte 1	Byte 2	Byte 3	Description
X'C1'	X'00'	A,I,S	not used	Irrecoverable machine check.
X'C2'	X'00'	A,I,S	not used	Irrecoverable channel check during FETCH.
X'C3'	X'00'	A,I,S	not used	Irrecoverable channel check on paging channel.
X'C5'	X'00'	A,I,S	not used	No ECSW stored.
X'C7'	X'00'	A,I,S	not used	Channel failure; channel address invalid.
X'C8'	X'00'	A,I,S	not used	Channel failure on SYSLOG

#### Notes:

- A** X'C1' - SYSREC recording unsuccessful (No record written)
- I** X'C9' - SYSREC recording incomplete (Not all records written)
- S** X'E2' - SYSREC recording successfully completed

Figure 169. MCH/CCH Wait Codes

### Device Error Recovery Wait Codes

Byte 0	Byte 1	Byte 2	Byte 3	Description
X'08' to X'60'	X'C1' or X'C4'	Channel	Unit	Error recovery messages.

Figure 170. Device Error Recovery Wait Codes. For Error Recovery Messages refer to **0P...** messages in z/VSE Messages and Codes, SC33-8306/8307/8308.

### SDAID Soft Wait Codes

Byte 0	Byte 1	Byte 2	Byte 3	Description
X'62'	X'C5'	Not used	Not Used	SDAID output device became unready. Make printer ready and press the EXTERNAL INTERRUPT key.
X'00'	X'00'	X'00'	X'00'	SDAID stop on event. To continue, press the EXTERNAL INTERRUPT key.

Figure 171. SDAID Soft Wait Code. (Identified by EEEE in the address part of the WAIT PSW).

## IPL Hard Wait Codes

Byte 0	Byte 1	Byte 2	Byte 3	Explanation
X'07'	X'E6'	Device number		IPL input/output error: - The identified device is preventing IPL to complete its process.
X'07'	X'E6'	X'C3'	X'E2'	Console router error 'CS': Byte 4,5: 2 bytes return code Byte 6,7: 2 bytes error code <sup>2</sup>
X'07'	X'E6'	X'C9'	X'C3'	Integrated Console error 'IC': Byte 5: 1 byte return code Byte 6,7: 2 bytes error code <sup>2</sup>
X'C1'	X'E2'	not used	not used	Unrecoverable machine check
X'cc'	X'00'	X'0F'	X'D0'	Error during IPL. IPL canceled. (cc=cancel code) <sup>3</sup>
X'F0'	X'C9'	X'F0'	X'F0'	See message 0I00
X'F0'	X'C9'	X'F0'	X'F1'	See message 0I01
X'F0'	X'C9'	X'F0'	X'F6'	The device type of SYSRES can not be identified. The volume label (VOL1) or format-4 record contains invalid information. The pack was not initialized correctly.
X'F0'	X'C9'	X'F0'	X'F7'	See message 0I07
X'F0'	X'C9'	X'F1'	X'F4'	Unexpected return from SCLP See message 0I14
X'F0'	X'C9'	X'F5'	X'F4'	Phase not found, phase name is appended. See message 0I54
X'F0'	X'C9'	X'F6'	X'F8'	Unsupported Hardware. See message 0I68, RC=2
X'F0'	X'D1'	X'F1'	X'F7'	Too many devices in IOCDs. See message 0J17
X'F0'	X'D1'	X'F5'	X'F0'	Unsupported SYSLOG device. See message 0J5* sup4.

Figure 172. IPL Hard Wait Codes. For IPL Wait State Messages in low core refer to z/VSE Messages and Codes, SC33-8306/8307/8308.



## Hardwait Originator Codes for IPL

The Hardwait Originator Codes for Re-IPL can be found in the SYSCOM field IJBIPORG.

HWIPL000 EQU	0	UNKNOWN HARDWAIT ORIGINATOR
HWIPL004 EQU	4	RE-IPL CODE OR DATA DESTROYED
HWIPL008 EQU	8	SYSTEM IS NOT UNATTENDED
HWIPL012 EQU	12	ERROR OCCURRED DURING RE-IPL INIT
HWIPL016 EQU	16	I/O ERROR ON READ COUNTER RECORD
HWIPL020 EQU	20	COUNTER RECORD ON DASD DESTROYED
HWIPL024 EQU	24	NO RE-IPL (SET BY AR CMD AUTOIPL)
HWIPL028 EQU	28	SAME REASON LIMIT IS ZERO
HWIPL032 EQU	32	TOTAL IPL COUNTER LIMIT REACHED
HWIPL036 EQU	36	NO ALTERNATE IPL DEVICE SPECIFIED
HWIPL040 EQU	40	SAME REASON LIMIT REACHED AND
*		ALREADY ON ALTERNATE DEVICE
HWIPL044 EQU	44	SAME REASON LIMIT REACHED, SWITCH
*		NOT ALLOWED,DEVICE=PRIMARY REQUEST.
HWIPL048 EQU	48	BACK SWITCH TO CURR.DEVICE REJECT
HWIPL052 EQU	52	ALT. IPL DEVICE,SAME REASON LIMIT
*		REACHED, SWITCH TO PRIM. NOT ALLOW
HWIPL056 EQU	56	I/O ERROR ON WRITE COUNTER RECORD
HWIPL060 EQU	60	CP IPL FAILED
HWIPL068 EQU	68	I/O ERROR ON READ BOOTSTRAP
HWIPL072 EQU	72	MACHINE CHECK DURING RE-IPL CODE
HWIPL076 EQU	76	ERRONEOUS SVC OR EXTERNAL OR I/O
*		INTERRUPTION DURING RE-IPL CODE
HWIPL080 EQU	80	PROGRAM INTERRUPTION " " "
HWIPL084 EQU	84	CLOCK NOT OPERATIONAL

## Supervisor Hard Wait Codes

Byte 0	Byte 1	Byte 2	Byte 3	Description
X'00'	X'00'	X'0F'	X'EC'	Unexpected System Task cancelation
X'00'	X'00'	X'0F'	X'ED'	System error condition (e.g.control block inconsistency). General Register 5 contains the address of the location where the System inconsistency was determined.
X'00'	X'00'	X'0F'	X'F1'	System error detected by the page manager.
X'00'	X'00'	X'0F'	X'F2'	Unused
X'00'	X'00'	X'0F'	X'F3'	Unused
X'00'	X'00'	X'0F'	X'F4'	Unused
X'00'	X'00'	X'0F'	X'F5'	TFIX count outside limits.
X'00'	X'00'	X'0F'	X'F6'	I/O error during update of the SLD.
X'00'	X'00'	X'0F'	X'F7'	No copy blocks available for BTAM appendage I/O request.
X'00'	X'00'	X'0F'	X'F8'	Unused
X'00'	X'00'	X'0F'	X'F9'	Paging I/O error.
X'00'	X'00'	X'0F'	X'FA'	Translation specification exception.
X'00'	X'00'	X'0F'	X'FB'	Page fault in supervisor routine with identifier RID=X'00'.
X'00'	X'00'	X'0F'	X'FC'	Unused
X'00'	X'00'	X'0F'	X'FD'	Error during software Re-IPL
X'00'	X'00'	X'0F'	X'FE'	I/O error during fetch from SYSLIB.
X'00'	X'00'	X'0F'	X'FF'	Program check in supervisor or SDAID

**Note:** General Hard Wait Codes will be set by the VSE Supervisor or related routines at location x'00' to x'03' as shown above.

Usually the address field of the Hard Wait PSW is set to 00A00000 00001000. It is set to 00A00000 00001122 if a Program Check occurred within the Program Check Handler.

Figure 173. General Hard Wait Codes

## Hardwait Originator Codes for the Supervisor

The Hardwait Originator Codes for the supervisor can be found in the SYSCOM field IJBHWORG.

HWORG000	EQU	0		UNKNOWN HARDWAIT ORIGINATOR
HWORG004	EQU	4	SGEFCH	GETVCE OF FETCH-LIB DEVICE NOT OK
HWORG008	EQU	8	DISP	NO TASK FOUND IN SCHEDULED PART.
HWORG012	EQU	12	DISP	ERROR ON RESOURCE CHAINING
HWORG016	EQU	16	DISP	ERROR ON POSTING OF RESOURCE WAIT
HWORG020	EQU	20	SGNUC	SECOND SAVE AREA NOT ACCESSIBLE
HWORG024	EQU	24	SGNUC	GATING ROUTINES MISS SYSTEM QUEUE
HWORG028	EQU	28	SGPCK	CONTROL BLOCK INCONSISTENCY PCB
HWORG032	EQU	32	SGPCK	INCONSISTENCY PIB: PIK COMREG-PTR
HWORG036	EQU	36	SGPCK	PROGRAM CHECK IN SUPERVISOR
HWORG040	EQU	40	SGPCK	PAGE FAULT IN SUPERVISOR
HWORG044	EQU	44	SGPCK	TRANSLATION EXCEPTION
HWORG048	EQU	48	SGDFCH	FETCH I/O ERROR
HWORG052	EQU	52	SGCCWT	LRA FOR TFIXED ADDRESS NOT POSS.
HWORG056	EQU	56	SGCCWT	TFIX COUNT TOO HIGH
HWORG060	EQU	60	SGCCWT	NO COPYBLOCKS FOR 2ND TRANSLATION
HWORG064	EQU	64	SGSVC	CRT PHASE NOT FOUND, NO SL INFO.
HWORG068	EQU	68	SGSVC	CRT PHASE DOESN'T EXIST
HWORG072	EQU	72	SGSVC	CRT PHASE FETCH I/O ERROR
HWORG076	EQU	76	SGSVC	SYSTEM ERROR ON CRT PHASE
HWORG080	EQU	80	SGSVCX	TFIX COUNTER OVERFLOW
HWORG084	EQU	84	MCRAS	PERMANENT RAS I/O ERROR
HWORG088	EQU	88	MCRAS	RAS TRANSIENT FETCH ERROR
HWORG092	EQU	92	SGIOS	INCORRECT SVC0 ROUTINE RETCODE
HWORG096	EQU	96	SGIOS	INCORRECT OCCF ROUTINE RETCODE
HWORG100	EQU	100	SGIOS	EXTENT ENTRY DEQUEUE ERROR



## Cancel Code to Message Code Cross-Reference

Cancel Code (Hex)	Message Code	Descriptive Part of Message (or Condition)
00	—	In all cases default value except those listed
08	0V16I	CANCEL request from VSE/POWER
09	0V15I	CANCEL request from LIOCS
0A	0S21I	Processing error in access control
0B	0S20I	Access control violation
0C	0S19I	Execution failure in ICCF interactive partition
0D	0V13I	Program check in subsystem or appendage
0E	0V14I	Page fault in subsystem or appendage
0F	0P80I	Invalid 'read from/or write to' system file on FBA device
10	—	Normal EOJ
11	0V07I	No channel program translation for unsupported device
12	0V06I	Insufficient buffer space for channel program translation
13		Reserved
14	0V04I	Page pool too small
15	0V02I	Page fault in disabled program
16		Reserved
17	0S02I	(Same as 23 but causes dump because subtasks were attached when maintask issued CANCEL macro)
18	—	Eliminates cancel message when task issues DUMP macro
19	0P74I	I/O operator option
1A	0P73I	I/O Error
1B	0P82I	Channel failure
1C	0S14I	CANCEL ALL macro
1D	0S12I	Maintask termination
1E	0S13I	I/O error on lock file
1F	0P81I	CPU failure
20	0S03I	Program check
21	0S04I	Illegal SVC
22	0S05I	Phase not found
23	0S02I	Program request
24	0S01I	Operator intervention (cancel)
25	0P77I	Invalid address
26*	0P71I	SYSxxx not assigned (unassigned LUB code)

\* If the CCB/IORB is unavailable, the logical unit is SYSxxx.

Figure 174 (Part 1 of 2). Cancel Code to Message Code Cross-Reference

Cancel Code (Hex)	Message Code	Descriptive Part of Message (or Condition)
27	0P70I	Undefined logical unit (invalid LUB code in CCB)
28	0S35I	Phase too long (does not fit in LTA or partition)
29	0P92I	Invalid Sublibrary structure
2A	0V10I	I/O error on page data set
2B	0P84I	I/O error during fetch from private core image library
2C	0V09I	Illegal parameter passed by PHO routine
2D	0P88I	Failing storage block (program cannot be executed)
2E	0S16I	Invalid resource request (possible deadlock)
2F	0V03I	More than 255 PFI requests for 1 page
30	0P72I	Reading past /& statement (on SYSRDR or SYSIPT)
31		Reserved
32	0P76I	Invalid DASD address
33	0P79I	Invalid first CCW
34	0P93I	GETVIS space exhausted
35	0P85I	Job control open failure
36	0V08I	Program check or page fault in I/O appendage routine
37		Reserved
38	0V11I	Wrong privately translated CCW
39	0V12I	Invalid CCW chain for SYSLOG
3A	0V17I	Spool request out of sequence
3B	0V18I	VSE/OCCF detected error in canceled partition
3C	0V19I	VSE/OCCF subtask requested cancel of maintask
3D	L177I	Error during FETCH PFI processing
40	0V95I	VTAM error (termination of task)
41	0V96I	VTAM error (invalid condition code)
42	0P86I	Invalid dasd extent information
43	0P94I	Not possible to execute in dynamic partition
44	0S22I	Security manager error
45	0S17I	Execution mode violation
46	0S15I	Cancel code for data space services
47	0S11I	Cancel with abend code
48	0S27I	Cancel(0S/390 system completion code provided)
49	0S28I	0S/390 ABEND macro issued
4A	0S37I	Task canceled because service provider terminated
FF		Multiple cancel condition (see SYSLST for details)
xx	0P78I	Unrecognized cancel code
	0P83A*	Supervisor catalog failure
	0P87A*	IPL failure

\* This cancel code is not significant in case of a supervisor catalog or IPL failure, because the System is placed in a WAIT state without any further processing by the Terminator.

Figure 174 (Part 2 of 2). Cancel Code to Message Code Cross-Reference

---

# Debugging Facilities

All the debugging facilities described in this chapter must

**NOT BE TREATED AS AN INTERFACE OF ANY KIND.**

Any of the facilities might be subject to changes and/or additions whenever necessary.

## DEBUG Overview

This facility is written to debug the z/VSE Supervisor and will therefore be adapted to internal needs whenever necessary.

The debug facility can be enabled via the AR '**DEBUG ON**' command and disabled via the AR '**DEBUG OFF/END**' command.

A debug area will be allocated in the 31-bit system GETVIS-Area, as soon as the **DEBUG=ON** command is being issued.

---

## Features

The following features are available:

- Feature 1**      Event Tracing
- Feature 2**      Address Compare Stop

One low core field is used to point to Debug Control Header

**X'270'**            address of Debug Control Header.

The Debug Control Header consists of the following four fullwords:

- XXHEADER**      address of current Debug Trace-area Header. The layout of the Debug Trace-area Header will be described in detail later in this chapter.
- XXESINFO**      address of Event Selection control Information. The contents of the Event Selection control information is described in figure Figure 175 on page 412.
- Reserved**        This fullword is reserved for future extensions
- Reserved**        This fullword is reserved for future extensions

(pointed to by Debug Control Header (Word-1))

Bytes Dec	Bit	Corresponding Trace Entry written to Trace Area if Bit is 'on'
0 - 1	Trace event selection information	
1	0	EEEE00nn entries (program check entry) PCK
	1	Reserved for consistency check
	2	EEEE0200/0264 entries (display registers) REGS
	3	EEEE0300 entries (dispatcher exit) TASK
	4	EEEE0400 entries (I/O interrupt) INT
	5	EEEE0500 entries (start I/O) SIO
	6	EEEE0600 entries (external interrupt) EXT
	7	EEEE0700 entries (dispatcher entry) DISP
	8	EEEE0800 entries (supervisor call - SVC) SVC
	9	EEEE0900 entries (cancelation) TERM
	10	EEEE0A00 entries (switch trace area) SWCH
	11	EEEE0B00 entries (display data) DATA
	12	EEEE0C00 entries (user data) USER
	13	Reserved
	14	Reserved
2	15	EEEE0F00 entries (monitor call entry) MCL
	Debug Trace extention-1	
	0	Reserved
	1	Trace LOCK events LOCK
3	2	Trace TD events
	3	Trace BSM events BSM
	Debug Trace Extention-2	
	0	Page protection stop defined
3	1	page protection active
	Reserved	
4 -35	Reserved	
36-37	Usage count information	
	0	SVC usage count requested
	1	Fast SVC usage count requested
	2	Bound state usage count requested
	3	Performance counters requested
	4	SVC-117 counters requested
5	SVC-132 counters requested	
38-39	TID of monitored partition	
40-43	Monitoring buffer address	
44-95	Used internally	
96 -99	Partition Debug trace area address	
100-227	Address Compare Start/Stop information	
100-101	Space ID	
102	0-3	Processing bits
102	4-7	Number of bytes to be compared 0<n<9
103	mask byte to be used for the compare result	
104-107	address of field to be compared	
108-115	hex data string of comparand	
-----	-----	
116-227	7 Repeats of fields (byte 100-115)	

Figure 175. Event Selection Control Information



## Feature 1 - Event Tracing:

Trace entries are set up whenever a program check/page fault, external interrupt, I/O interrupt or SVC interrupt is encountered; or, in addition, whenever a SSCH instruction is issued, a task is being dispatched or whenever a task is being canceled. User trace entries will be generated whenever appropriate. All the trace entries are written into the Debug Trace Area.

Word-1 (byte 4-7) of the Debug Control Header contain the address of an Event Selection Parameter list, which contains in byte 0 to 1 a bit map where each bit represents a special trace event. This Event Selection Parameter List is used for **selective** debug traces. A bit of ON (1) indicates that the appropriate trace entry is active whereas a bit being OFF (0) indicates that the appropriate entry is inactive, which means no such trace entry will be written into the Debug Trace Area (see Figure 175 on page 412).

All the bits of the Selective Event Trace can be turned ON/OFF by means of the:

**DEBUG TRACE={ [NO]xxx, [NO]yy...., [NO]zzz }**

command, where xxx, yy, ... zzz are the options that the operator can choose. For details see the **DEBUG TRACE....** command later in this appendix.

## Feature 2 - Address Compare Stop:

The Debug Control Header contains special entries which contain information about when and under which conditions DEBUG is to Stop its event recording. There is a total of four entries and each of the entries contains the space-Id in byte 0-1 followed at offset 2 by the number of bytes ( $0 < n < 9$ ) that are to be compared followed at offset 3 by the mask byte representing the compare mask and at offset 4-7 the address of the field that, if its contents matches the contents given in bytes 8-15 of the appropriate Stop entry will lead to a **Hard Wait** next time the Debug processing routine is being entered. The Hard Wait PSW will be loaded before the new debug entry is being set up, which means that the event or task that caused the last Debug entry in the Debug-area to be created did cause the DEBUG STOP to occur subsequent to the creation, of this last Debug entry. For details see the **DEBUG STOP....** command later in this appendix. In addition see Figure 175 on page 412.

The PSW address portion will contain **0000EEEE** to identify the STOP condition. Pressing the Restart Key on the CPU, or specifying SYSTEM RESTART under VM, will cause the address compare stop to be automatically reset and normal processing sequence to continue as if no stop would have been encountered.

---

## How to Find and Read the Debug Trace Area

At fixed storage location **X'270'** you will find the address of the Debug Control Header. The Debug Control Header does contain the address of the currently active Debug Trace-area, or zero, if no debug information is available, at offset 0. Each Trace-area has in front a 16-byte header (4 words).

- Word zero points to the next free entry within this trace area.
- Word one points to the NEXT TRACE AREA which will be used when a cancel condition occurs or, when an exclusive switch request is being recognized.
- Word two points to the location within this trace area, where the last Debug-trace Entry can be build.
- Word three points to the PREVIOUS TRACE AREA that was used assuming it had been used at all.

word 0	word 1	word 2	word 3
next free entry	next area	end of area	previous area

*Figure 176. Debug-trace Area Header*

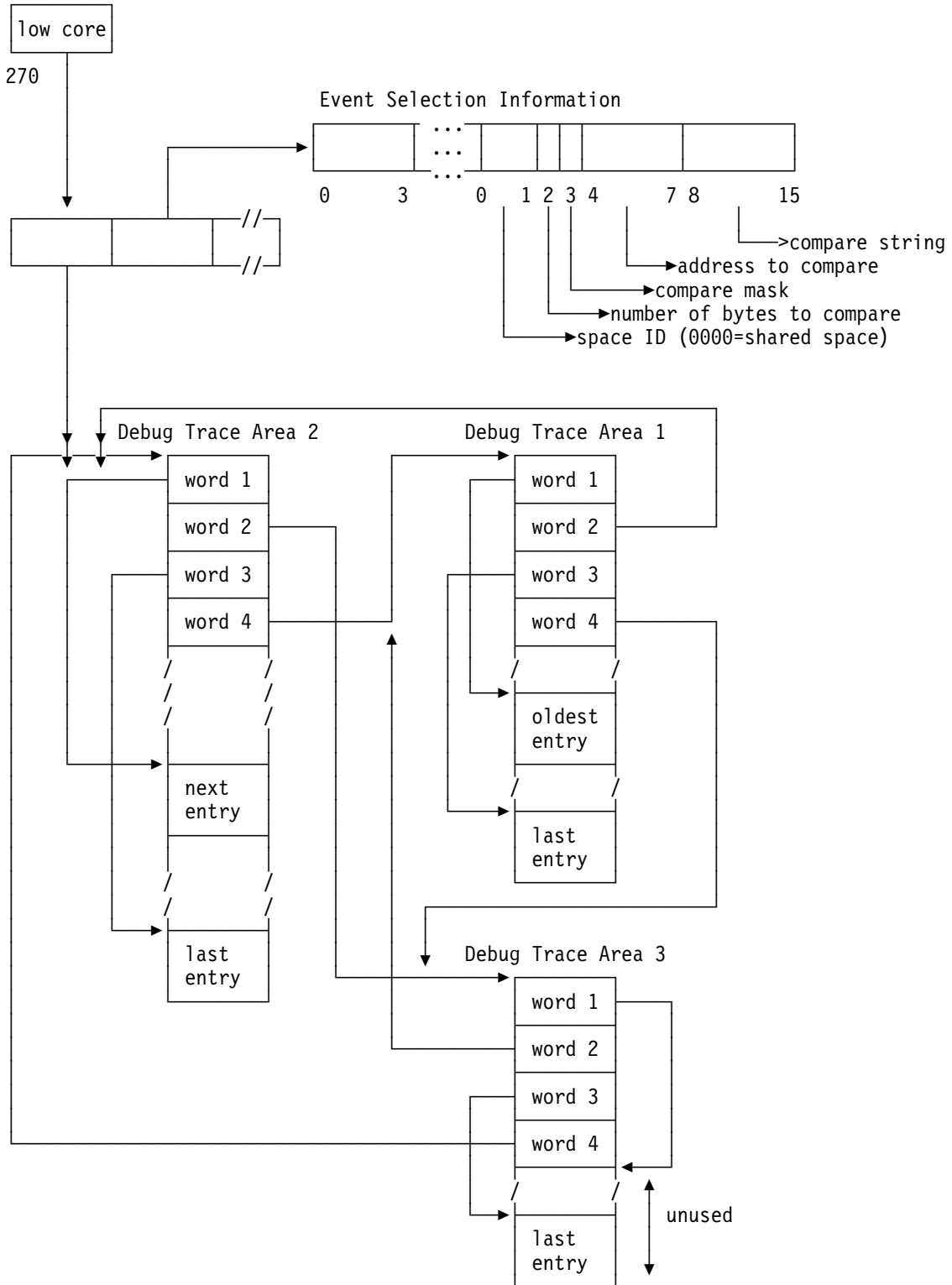


Figure 177. Relationship of Debug Areas

## Format of the Debug Trace Entries

The trace entries are always multiple of 16 bytes in length. The first two bytes of each entry contain **X'EEEE'** to simplify the recognition of the variable length trace entries.

### Program Check Entry (Param.: PCK - EEEE00ic)

Dec	Hex	
0- 3	0- 3	X'EEEE00ic' (ic is the program interruption code from location X'8F')
4-11	4- B	Program check old PSW
12-13	C- D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Program Check Interruption code (X'8C')
32-35	20-23	Transl.except.addr. for PC interrupt code x'10' and x'11' and x'14'
	or	
32-35	20-23	Break Event Address Register contents if PER3 facility installed
	or	
32-35	20-23	Contents of general register 0
36-95	24-5F	Contents of general register 1 trough 15

## Display All Registers (Param.: REGS - EEEE0200)

Dec	Hex	
0- 3	0- 3	X'EEEE0200'
4-11	4- B	Debug caller PSW
12-13	C- D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-95	20-5F	Contents of General Registers 0 to 15
	or	
28-31	1C-1F	Constant 'AREG'
32-95	20-5F	Contents of Access Registers 0 to 15
	or	
28-31	1C-1F	Constant 'CREG'
32-95	20-5F	Contents of Control Registers 0 to 15

**Note:** A second All Register Entry containing the contents of the Access Register will be appended to the the first All Register entry whenever the issuing program is running in Access Register mode. Such an entry does contain the character 'AREG' in byte 28-31 of the entry.

An All Register Entry containing the contents of the Control Registers will be generated prior to another trace entry in case the contents of one of the control registers has changed, e.g. through a space switch. Such an entry does contain the character 'CREG' in byte 28-31 of the entry.

Dec	Hex	
0- 3	0- 3	X'EEEE0264'
4-11	4- B	Debug caller PSW
12-13	C- D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-159	20-9F	Contents of 64-Bit General Registers 0 to 15
	or	
28-31	1C-1F	Constant 'AREG'
32-95	20-5F	Contents of Access Registers 0 to 15
	or	
28-31	1C-1F	Constant 'CREG'
32-95	20-5F	Contents of Control Registers 0 to 15

## Dispatcher Exit (Param.: TASK - EEEE0300)

Dec	Hex	
0- 3	0- 3	X'EEEE0300'
4-11	4- B	PSW of selected task
12-13	C- D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-95	20-5F	Contents of General Registers 0 to 15

**Note:** An All Register Entry containing the contents of the Control Registers may be generated prior to the Dispatcher exit entry in case the contents of one of the control Registers has changed. Such an entry does contain the character 'AREG' in byte 28-31 of the All Register entry. An All Register Entry containing the contents of the Access Register will be appended to this Dispatcher Exit entry whenever the selected program is running in Access Register mode. Such an entry does contain the character 'AREG' in byte 28-31 of the entry.

## I/O Interrupt (Param.: INT - EEEE0400)

Dec	Hex	
0- 3	0- 3	X'EEEE0400'
4-11	4- B	I/O check old PSW
12-13	C- D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Subchannel-Id word
32-33	20-21	Constant
34-35	22-23	Device-Id (cuu)
36-51	24-33	IRB
52-59	34-3B	CSW
60-63	3C-3F	CAW

## Start I/O (Param.: SIO - EEEE0500)

Dec	Hex	
0- 3	0- 3	X'EEEE0500'
4-11	4- B	Debug caller PSW
12-13	C- D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Subchannel-Id word
32-43	20-2B	ORB Operation Request Block
44-47	2C-2F	CHANQ information
48-51	30-33	Address of real CCB
52-55	34-37	Device-Id (cuu)
56-59	38-3B	Address of PUB entry
60-63	3C-3F	Address of CHANQ entry.

## External Interrupt (Param.: EXT - EEEE0600)

Dec	Hex	
0- 3	0- 3	X'EEEE0600'
4-11	4- B	External old PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-29	1C-1D	Reserved
30-31	1E-1F	External Interruption code (location X'86') if External interruption code = 1xxx
32-39	20-27	Clock Comparator
40-47	28-2F	CPU Timer
	or	
32-63	20-3F	if External interruption code = 4xxx External Interrupt buffer information (APPC)

## Task Selection (Param.: DISP - EEEE0700)

Dec	Hex	
0- 3	0- 3	X'EEEE0700'
4-11	4- B	Debug caller PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-33	20-21	LTID (task ID of task owning the LTA)
34-35	22-23	Mnemonic partition ID
36-39	24-27	System task working for task if any
40-43	28-2B	TIB-address
44-47	2C-2F	TIBSCB
48-49	30-31	TIBRTID
50-51	32-33	TIBRPIK
52-53	34-35	TIBRQPIK
54	36	TIBFLAG1
55	37	TIBFLAG
56	38	TIBFLAG2
57	39	TIBCACL
58	3A	TIBCACL2
59	3B	TIBCACL3
60	3C	TIBFLAG3
61	3D	TIBDMFLG
62	3E	PCEFLAG
63	3F	PCEFLAG1



## Supervisor Call (Param.: SVC - EEEE0800)

Dec	Hex	
0- 3	0- 3	X'EEEE0800'
4-11	4- B	Supervisor call old PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-29	1C-1D	Constant
30-31	1E-1F	SVC interruption code in the form xxnn xx, if unequal to 00 is the OS/390 SVC code
32-48	20-2F	Contents of General Registers 15 to 2

## Cancel Entry (Param.: TERM - EEEE0900)

Used at ERRGO.

Dec	Hex	
0- 3	0- 3	X'EEEE0900'
4-11	4- B	Debug caller PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-33	20-21	LTID (task ID of task owning the LTA)
34-35	22-23	Mnemonic partition ID
36-39	24-27	System task working for task if any
40-43	28-2B	TIB-address
44-47	2C-2F	TIBSCB
48-49	30-31	TIBRTID
50-51	32-33	TIBRPIK
52-53	34-35	TIBRQPIK
54	36	TIBFLAG1
55	37	TIBFLAG
56	38	TIBFLAG2
57	39	TIBCNC1
58	3A	TIBCNC2
59	3B	TIBCNC3
60	3C	TIBFLAG3
61	3D	TIBDMFLG
62	3E	PCEFLAG
63	3F	PCEFLAG1

## Switch Debug Trace Area (Param.: SWCH - EEEE0A00)

This Debug call forces a Debug Trace Area swap (done after cancelation or Debug was set ON again).

Dec	Hex	
0- 3	0- 3	X'EEEE0A00'
4-11	4- B	Debug caller PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-39	20-27	Phase name currently in LTA
40-47	28-2F	Phase name currently in PTA

## Display Data (Param.: DATA - EEEE0B00)

Before issuing the Debug call, the address of the data to be saved must be stored to XXUSAREA in low core.

Dec	Hex	
0- 3	0- 3	X'EEEE0B00'
4-11	4- B	Debug caller PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-47	20-2F	Data, addressed by the address in XXUSAREA

## Display User Data (Param.: USER - EEEE0C00)

This Debug entry will cause from one to a max. of four data areas, each with a fixed length of 16 bytes, to be saved in the Debug Trace Entry. The data must be in storage when the debug call is being issued.

Dec	Hex	
0- 3	0- 3	X'EEEE0C00'
4-11	4- B	Debug caller PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Reserved
32-47	20-2F	Data provided by Debug call address-1 optional
48-63	30-3F	Data provided by Debug call address-2 optional
64-79	40-4F	Data provided by Debug call address-3 optional
80-95	50-5F	Data provided by Debug call address-4

## Monitor Call Entries (Param.: EEEE0Fnn)

Dec	Hex	
0- 3	0- 3	X'EEEE0Fnn' (nn is the monitor call class)
4-11	4- B	Monitor call PSW
12-13	C-D	TID (task ID)
14	E	RID (routine ID)
15	F	CPU state
16-23	10-17	Time Stamp
24-25	18-19	CPU-Id
26-27	1A-1B	Reserved
28-31	1C-1F	Program Check Interruption Code (X'8C')

---

## DEBUG ON Command

### Command Description

DEBUG [{ON[,nnnK]|OFF|END}]

- DEBUG ON(,nnnK)** Three areas in the specified length, default is 16K bytes each, will be allocated within the 31-bit System GETVIS Area and one area at a time will be used in wrap around mode to hold the trace information. An area swap will automatically be made whenever a task ends abnormally (is being canceled).  
The area swap will also be made when debug is reactivated after temporary deactivation (DEBUG OFF).
- DEBUG id** A single Debug area of 16K bytes will be allocated within the 31-bit System GETVIS Area and it will be used in wrap around mode to hold the trace information of the specified partition only (id=BG,F1,F2,...Fn) This trace area will be maintained IN ADDITION to the commonly used DEBUG trace-areas.
- DEBUG OFF** The 'DEBUG OFF' command will suspend tracing of all debug call events until reactivated again (DEBUG ON).
- DEBUG END** The DEBUG END command causes tracing to be stopped immediately and all System GETVIS Areas allocated for Debug to be freed again.
- DEBUG** The 'DEBUG' command, if entered without any operands will cause the current TRACE setting to be displayed.

---

## DEBUG TRACE Command

### Command Description

DEBUG TRACE={option|option,option...[,option]}

- option** Specify the trace events that the user wants to be selected for tracing. Any of the following options can be chosen or suppressed.
- (NO)PCK** The user wants the PROGRAM-CHECK event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever a program check, excluding the program check resulting from an MC-instruction, is being recognized by the system.
  - (NO)REGS** The user wants the ALL-REGISTER event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever the contents of the general register seem of special interest to the user (e.g. in case of a Machine Check, or after completion of a FAST-SVC (6B) request).
  - (NO)TASK** The user wants the TASK-DISPATCHING event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever a task is being dispatched.
  - (NO)INT** The user wants the I/O-INTERRUPT event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever an I/O interrupt has been recognized.
  - (NO)SIO** The user wants the SSCH event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever the I/O scheduler has initiated an I/O operation.
  - (NO)EXT** The user wants the EXTERNAL-INTERRUPT event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever an external interrupt has been recognized.
  - (NO)DISP** The user wants the TASK-SELECTION event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever the dispatcher is being entered for task selection.
  - (NO)SVC** The user wants the SUPERVISOR-CALL event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever an SVC has been issued.
  - (NO)TERM** The user wants the ABNORMAL-TERMINATION event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever a task is being canceled for whichever reason. As a result of the termination, a switch to the next DEBUG Trace Area will be made, assuming the appropriate event selection bit (SWCH) has not been set OFF.
  - (NO)SWCH** The user wants the SWITCH-DEBUG-AREA event to be traced (or suppressed in case the NO option was chosen). These type of events will occur after a task has been canceled, or after DEBUG has been re-activated (DEBUG ON). If the user chooses NOSWCH, no switching to the next DEBUG Trace Area will be initiated and the current area remains active.

- (NO)DATA** The user wants the SPECIAL-DATA event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever some data seems of special interest to the user.
- (NO)USER** The user wants the USER-DATA event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever user data seems of special interest to the user. This trace event is thought to be used for user convenience.
- (NO)MCL** The user wants MONITOR-CALL event to be traced (or suppressed in case the NO option was chosen). These type of events will occur whenever a monitor call is being recognized by the system.
- NONE** The user wants ALL trace events to be set OFF.
- ALL** The user wants ALL trace events to be set ON.

The options can be set in any order and must be separated by a comma.

**Note:** The default setting is:

DEBUG TRACE=PCK,REGS,TASK,INT,SIO,EXT,DISP,SVC,TERM,SWCH,DATA,USER

---

## DEBUG SELECT Command

### Command Description

```
DEBUG SELECT,{INT=cuu|SIO=cuu|SVC=no}
```

This command will cause selected entries to be traced. Selective entries will only be provided for I/O (SIO, INT) and Supervisor call (SVC) events. A single device (cuu) or SVC (number) can only be provided per DEBUG SELECT command, however, more than one command can be given. A total of 32 devices and all Supervisor calls can be selected. In order to reset the selected events, DEBUG must first be turned OFF and then turned ON again.

---

## DEBUG SHOW Command

### Command Description

```
DEBUG [P|N]SHOW{=option|option,option...[,option][,CUU=cuu]}
```

This command will cause the (selected) entries in the (specified) Debug Trace Area to be formatted properly and to either be displayed on the system operator console, or printed on a specified printer.

**PSHOW** The user wants the **previous** Debug Trace Area to be displayed on the console (or printed on the specified printer).

**NSHOW** The user wants the **next** Debug Trace Area to be displayed on the console (or printed on the specified printer).

**Note:** This area, if it contains data at all, does contain the oldest trace entries that are available within all Debug Trace Areas.

**SHOW** The user wants the **current** Debug Trace Area to be displayed on the console (or printed on the specified printer). No new Debug Trace entries will be generated as long as printing the currently active Debug Trace Area is ongoing.

**Note:** It is recommended to suspend debug (DEBUG OFF) before using the DEBUG SHOW.... command.

**SHOW=id** The user wants the **Partition** Debug Trace Area to be displayed on the console (or printed on the specified printer).

**options** Specify the entries which are to be SELECTED (or EXCLUDED) for displaying on the console (or printing on the specified printer). The options are the same as specified in the DEBUG TRACE command (see **DEBUG TRACE...** command in the previous section).

**cuu** Specifies the printer on which the output is to be printed. This operand, if it has been omitted, causes the trace entries to be displayed on the operator console.

To get all three Debug Trace Areas the operator has to issue all three commands.



---

## DEBUG STOP Command

### Command Description

DEBUG STOP condition[,{AND|OR},condition]

condition = [id,][+]addr.len,{EQ|NE|LO|HI},string

This command will cause some data at a user specified address to be compared to a string of user specified data. If this compare matches all the condition which where specified by the user, the system will cause a Hard Wait PSW to be loaded. It should be noted, however, that the data compare will ONLY take place whenever a Debug call is being attempted, regardless of whether the appropriate trace event is enabled or disabled.

The address of the WAIT PSW will contain 0000EEEE which uniquely identifies a MATCH STOP condition. Activating the PSW RESTART key, or, in case you are running under VM, entering SYSTEM RESTART will cause normal operation to continue.

**Note:** The DEBUG STOP condition will be reset once it has been recognized. To stop another time, the user must reenter the same command

**STOP** the user wants to **STOP** processing as soon as a specified condition is being recognized by the Debug Processing routine.

**id** specifies the address space, or specifies the SYSLOG-Id of the partition, whose address is to be searched for a matching string.

**+** indicates that the address is relative to the begin address of the specified partition. The system will ADD the partition start address to the given relative **addr.** to form an absolute address.

**addr** is the address [relative if + is prefixed to the address] that needs to be inspected.

**len** is the number of bytes  $0 < n < 9$  that the user wants to be compared. This number must match the number of bytes specified in the **string** operand.

**EQ|NE|LO|HI** is the operator which is to be applied for the comparison operation.

EQ the operands must be equal

NE the operands must be unequal

LO the data at the given address must be lower than the string

HI the data at the given address must be higher than the string

**string** is a hex-character string representing the data that the storage contents is to be compared to.

**AND|OR** specifies the additional condition that must be satisfied for the Address Compare Stop

AND both specified conditions must be met before the system stops

OR either of the two conditions, if met will cause the system stop.

---

## LOCATE Command

### Command Description

The LOCATE command scans the virtual storage for the next occurrence of either a character-, or a hexadecimal-character string whereby parts of the string may be unknown. The total string is limited to 16 characters or 32-hex digits.

The format of the command is as follows:

```
LOCATE [id,][']string [[FROM=start][,TO=end][,RUN]]
```

**id** specifies the SPACE or the PARTITION which is to be scanned for the specified **string**.

If **id** is omitted, the space will be defaulted to private space number 0.  
Valid IDs are:

S SHARED space

R REAL space

1 Private space number one

2 Private space number two

n Private space number n

BG BG (Background) partition

F1 F1 (Foreground) partition one

F2 F2 (Foreground) partition two

.....

FB FB (Foreground) partition eleven

xy ID of dynamic partition

**'** is the single, special character that must be used to indicate that the following string is a **character string**.

If the **'** is missing, the string is assumed to be hexadecimal digits.

**string** is the string that the user wants to be located; limited to either 16 characters or, to 32 hexadecimal digits representing 16 bytes of storage.

Any character or hexadecimal digit that should be **excluded** from the scan, must be presented by a **.** (dot).

An even number of hexadecimal digits, including **.** (dots), must be specified in case of hexadecimal digit scan.

## OPTIONS

Any additional processing option(s) that the user wants to specify must be separated from each other by a colon and the first OPTION specified must be preceded by a left parenthesis. The following processing options can be specified:

- FROM=start** specifies an address within the specified **id** where the scan should begin. The FROM option, if omitted will force the scan to be started at the first byte of the given **id**. The user may want to specify an offset within the specified **id** rather than an absolute address. In this case the **offset** must immediately be preceded by a **+**.
- TO=end** specifies the address within the specified **id** where the scan should end. The TO option, if omitted will force the scan to be ended at the last byte of the given **id**. The user may want to specify an **offset** or a **length** rather than an absolute address to indicate where the scan should be ended within the specified **id**. In this case an **offset** must be indicated by a **+** immediately preceding the offset relative to the begin of the specified **id**, whereas a length can be identified by a leading **.** (dot).
- RUN** indicates that all addresses where a match is found (within the calculated or given boundaries) should be logged onto the console without prompting for an operator response. The RUN option if omitted will cause the system to display 64 bytes of information starting at the next lower 16-byte boundary which precedes the next subsequent **string** match. The system will then WAIT for an operator response. A NULL response (ENTER) will cause the system to check for the **next** occurrence of a matching **string**. This will be repeated until a NONULL response is received, or, until the scan boundaries are reached. Operator prompting mode is the default option.

The LOCATE function can be terminated by entering "15 END".

---

## SHOW Command

### Command Description

The SHOW command displays a defined number of bytes from a given address in the specified space on the operator console. The virtual address as well as its related real address, if applicable, will be displayed on the console.

The format of the command is as follows:

```
SHOW [{id|id,DSPNAME=name|id,SGT|id,PT},]address[.length]
```

**id** specifies the SPACE or the PARTITION the bytes of which are to be displayed on the console.

If **id** is omitted, the space will be defaulted to private space number 1.

Valid IDs are:

S SHARED space

R REAL space

1 Private space number one

2 Private space number two

n Private space number n

BG BG (Background) partition

F1 F1 (Foreground) partition one

F2 F2 (Foreground) partition two

.....

FB FB (Foreground) partition eleven

xy ID of dynamic partition

**DSPNAME=name** name is the name of a data space which belongs to the specified partition.

**SGT** indicates that the system is to provide segment table information for the specified space or partition and the eventually given **address**.

**PT** indicates that the system is to provide page table information for the specified space or partition and the eventually given **address**.

**address** is the begin address of the area that is to be displayed, or for which **SGT** or **PT** information is to be provided, on the console once the associated space addressing has been established. The address can be in the range from 1 to 6 hexadecimal digits.

#### OPTIONS

The user can append a length specification to the address which must be separated from the address by a . (**dot**) and the length must not exceed 4095 bytes.

This publication contains appendixes as follows:

- Appendix A: Supervisor Data Areas (without I/O)
- Appendix B: I/O Control Blocks supported by VSE system.
- Appendix C: Samples  
Track hold processing examples.
- Appendix D: XPCP / APPCVM Protocol
- Appendix E: Performance Monitoring Interface



---

## Appendix A. Supervisor Data Areas (without I/O)

Below you find an overview of the most important supervisor data areas.

For each data area there is a short description of the data area and how you can address it.

There is also a reference to the mapping macro for the data area. The name of the mapping macro can be used to find the layout (DSECT) of the data area in the supervisor listing.

You also find references to figures which show the interrelationships between data areas.

For an overview of the I/O Control Blocks see Appendix B.

*Figure 178 (Page 1 of 3). Overview of the Supervisor Data Areas (except I/O data areas)*

<b>Data Area</b>	<b>Name of Data Area</b>	<b>Addressability</b>	<b>Mapping Macro</b>
ACCTCOMN	Job Accounting Common Table contains informations about the Job Accounting Partition tables and job accounting areas	SYSCOM field IJBATAB	none
ACCTABLE	Job Accounting Partition Table contains accounting information for the partition	COMREG field JAPART	none
CLIM	Dynamic Class and System Limits Table describes system limits f.ex. for the partition size, the number of LUBs, the number of tasks and the number of partitions	SYSCOM field IJBCLIM Figure 179 on page 439	MAPCLIM
CLTAB	Dynamic Class Table one entry for each dynamic class; describes the attributes of the dynamic class f.ex. limits for the allocation of dynamic partitions, max. number of partitions within the class	SYSCOM field IJBCLTAB Figure 180 on page 440 Figure 181 on page 441	MAPCLASS
COMREG	Partition Communication Region one for each static and dynamic partition; contains information related to jobs running in the partition f.ex. job name, job start time, pointer to job accounting table	PCE field PCECOMRA Figure 183 on page 444 Figure 184 on page 446	MAPCOMR
CPCB	Dynamic Class Control Block one for each dynamic class; describes the dispatching control information for the dynamic partitions of this class	SYSCOM field IJBCLTAB Figure 181 on page 441 Figure 180 on page 440	MAPPCB
LOCKTAB	LOCK Table an entry contains the resource name, a pointer to the owner elements and a pointer to the request elements	ALOKTABA in supervisor Figure 191 on page 453	LOCKADR
PCB	Partition Control Block one for each static and dynamic partition; contains partition related information	GETFLD FIELD=PCBPTR Figure 184 on page 446	MAPPCB



Figure 178 (Page 2 of 3). Overview of the Supervisor Data Areas (except I/O data areas)

<b>Data Area</b>	<b>Name of Data Area</b>	<b>Addressability</b>	<b>Mapping Macro</b>
PCBATAB	Partition Control Block Address Table contains pointers to the partition control blocks (PCBs)	Low Core field x'264' Figure 184 on page 446	none
PCBX	Partition Control Block Extension 31-bit getvis extension of the PCB	PCB field PCBAPCBX	MAPPCCBX
PCE	Partition Control Block Extension one for each static and dynamic partition; contains partition related information	GETFLD FIELD=PCEPTR Figure 184 on page 446	MAPPCE
PCEATAB	Partition Control Block Extension Address Table contains pointers to the partition control block extensions (PCEs)	GETFLD FIELD=PCEATAB Figure 184 on page 446	none
PIB	Partition Information Block contains information about the partition status (inactive, stopped, ..) and the number of System and programmer LUBs	GETFLD FIELD=PIB Figure 183 on page 444 Figure 184 on page 446	MAPPPIB
PIB2	Partition Information Block Extension contains the TID of the main task, the PIK of the partition and points to the PCB	GETFLD FIELD=PIB Figure 183 on page 444 Figure 184 on page 446	MAPPPIB
PJB	Power Job Information Control Block describes VSE/POWER and VSE/AF job information f.ex. VSE/POWER job name, VSE/POWER job start time	PCE field PCEPOWJB	MAPPOWJB
SCB	Space Control Block describes the type and layout of an address space  shows the partitions running in the address space	Figure 182 on page 443	MAPSCB
SYSCOM	System Communication Region contains pointers to other supervisor areas, flag bytes, and system variables f.ex. number of partitions, number of channel queue entries, end of real storage.	ASYSKOM	SYSCOM
TCB	Task Control Block contains information about system and user tasks	GETFLD FIELD=TCBPTR Figure 185 on page 448	MAPTCB
TCBX	Task Control Block Extension 31-bit getvis extension of the TCB	GETFLD FIELD=TCBPTR Figure 185 on page 448	MAPTCBX
TIB	Task Information Block contains all task related information which has to be kept in fixed storage	TIBATAB Figure 185 on page 448	MAPTIB

Figure 178 (Page 3 of 3). Overview of the Supervisor Data Areas (except I/O data areas)

<b>Data Area</b>	<b>Name of Data Area</b>	<b>Addressability</b>	<b>Mapping Macro</b>
TIBATAB	Task Information Block Address Table contains pointers to the Task Information Blocks	Low Core x'2C0' Figure 185 on page 448	none
VIO Tables	VIO tables; see also SVC 114 (x'72')	SYSCOM field IJBVIOCM points to the VIO communication area Figure 196 on page 456	none

## Dynamic Class and System Limits Table (CLIMADR)

The 'CLIM' control block shows dynamic class and system limits.

SYSCOM

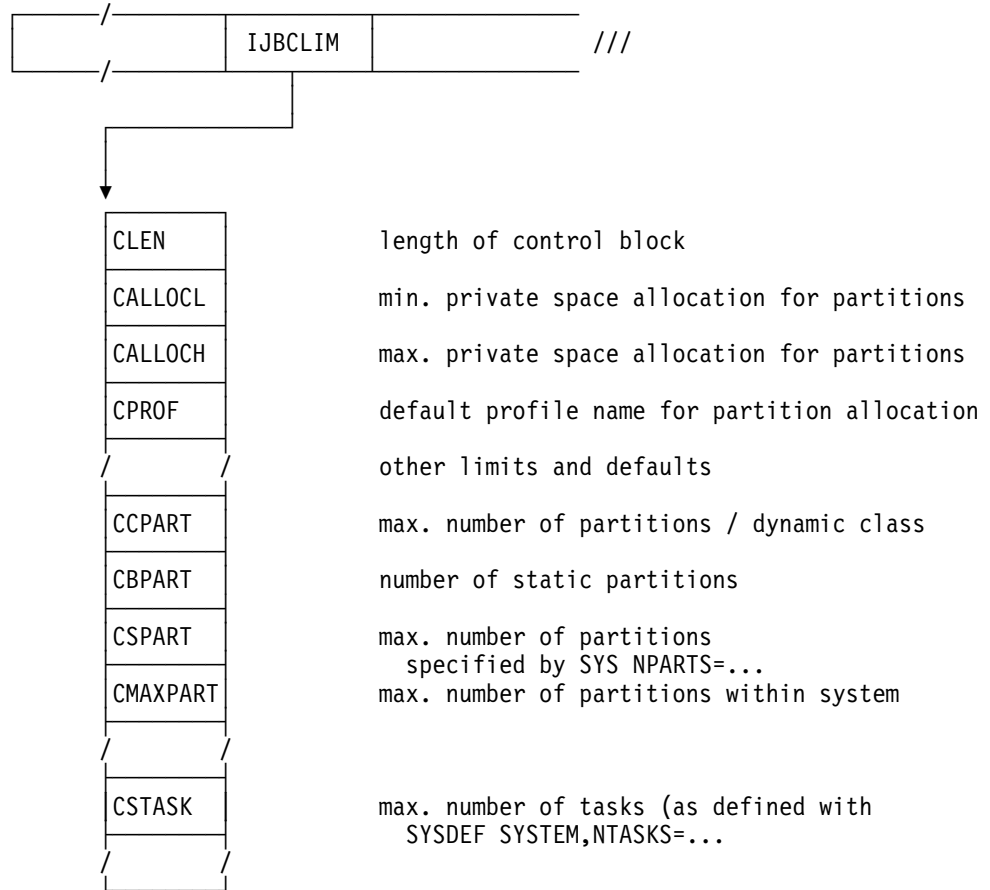


Figure 179. Dynamic Class and System Limits (CLIM) Relationship

## Dynamic Class Table

The dynamic class table contains attributes for dynamic classes and is stored as the member DTR\$DYNC.Z in the VSE/AF Library IJSYSRS.SYSLIB. The member shows the external representation of the dynamic class table. The VSE/POWER PLOAD command loads the the dynamic class table from the library, checks the contents for correctness and translates the member contents into the internal format. The PLOAD command process calls the DYNCLASS ID=LOAD service to activate and allocate if not already done the dynamic class table. The service connects one CPCB to each valid dynamic class table entry.

The following figure shows the interrelationship between the internal dynamic class table and the dynamic class control block (refer to Figure 180).

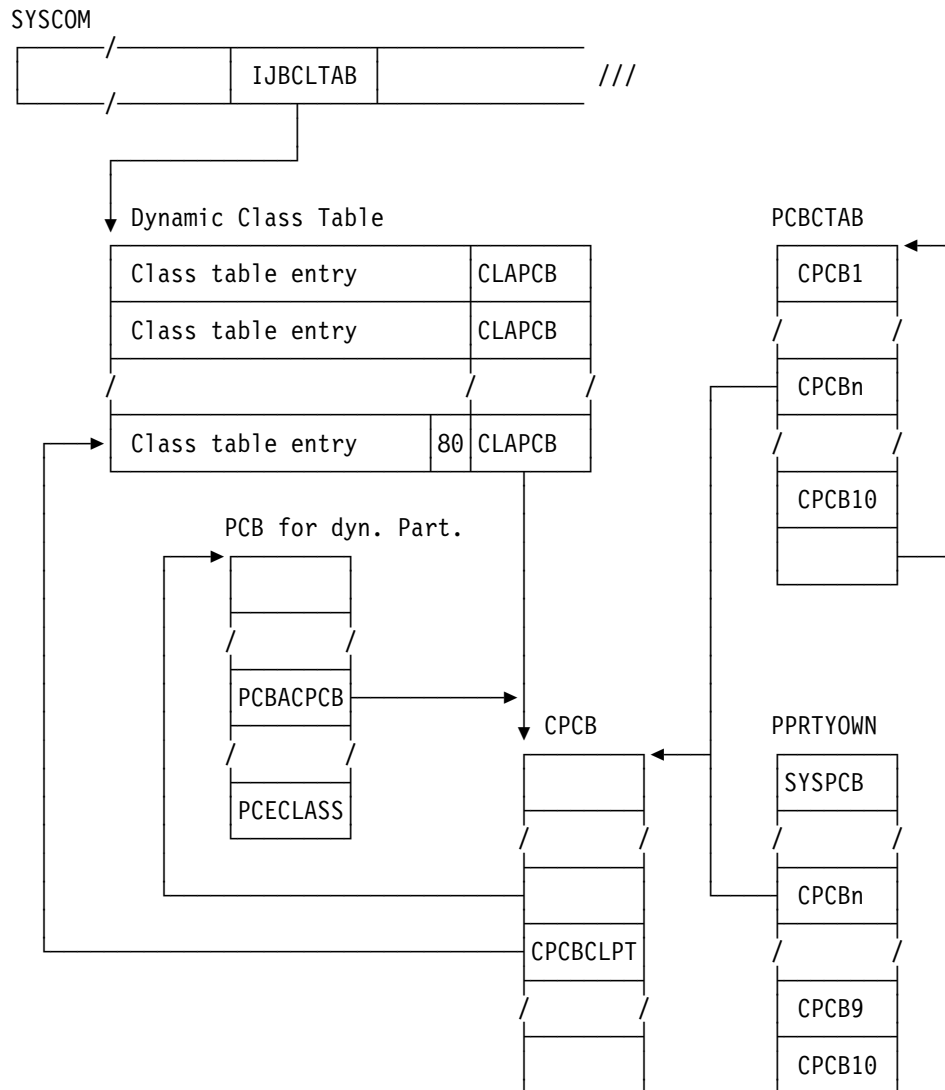


Figure 180. Dynamic Class Table - CPCB Interrelationship

## Dynamic Class Control Block (CPCBADR)

The dynamic class control block (CPCB) describes the dispatching control information and gives access to the attributes of a dynamic class. When the dynamic class character field (CPCBCLSS) of the CPCB contains X'FF', the CPCB is not connected to a dynamic class table entry, otherwise this field gives the character of the allocated dynamic class.

The CPCB contains

- the dynamic partition selection string (CPCPSS),
- CPCBCBPT points to the PCBs of the dynamic partitions in priority order, CPCBPLOW points to the lowest priority PCB address pointer
- control and CPU timer information.

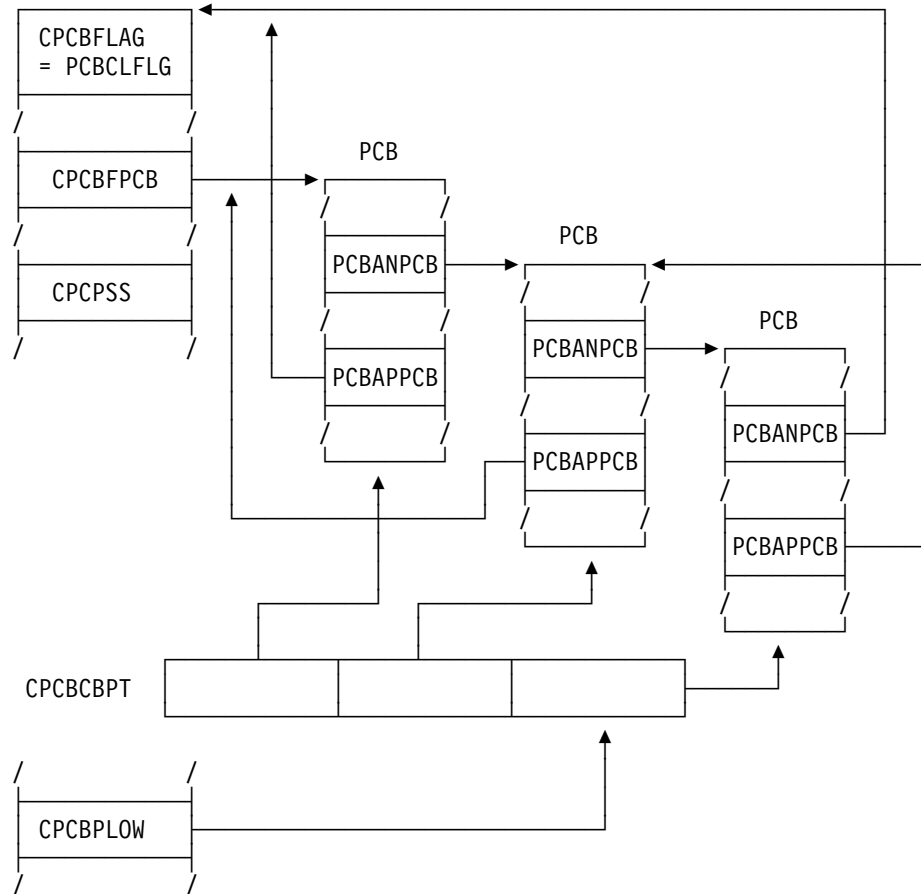


Figure 181. Dynamic Class Control Block - PCB Interrelationship

## Space Control Block (SCB)

A real/static space is allocated, when the first real/static partition within the space is allocated and deallocated when the last real/static partition within the space is deallocated.

Furthermore one dynamic space is allocated for each dynamic partition for which a POWER job is scheduled. This space is deallocated after the POWER job has finished.

Additionally there are page manager address spaces that contain the PMR tables for the allocated spaces (see "Segment-, Page-Table, PTAS and POSL" on page 196).

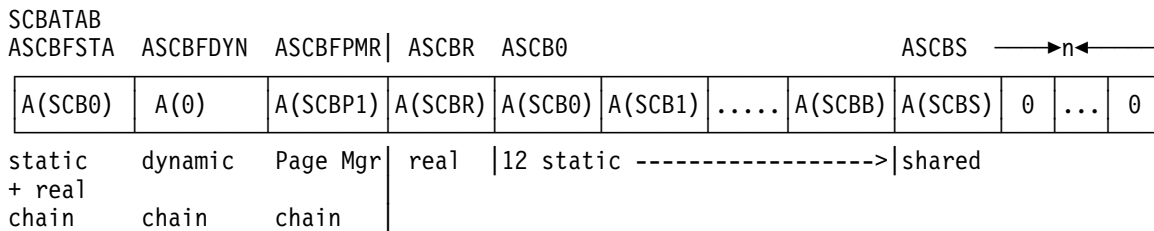
There is one shared address space. Within this space only the shared areas are valid. The private area is invalid.

Each address space is described by a SCB.

The SCBs for the real, the shared and the static spaces, as well as for the first PMR space are allocated during supervisor generation. SCBs for dynamic spaces and page manager spaces are allocated dynamically and freed when the space is deallocated. Currently page manager spaces are never deallocated. The pointers to the different SCBs, except page manager space SCBs, are contained in SCBATAB. The length of the SCBATAB is calculated depending on the SYS NPARTS= ... specification; there is one entry reserved for each dynamic space.

The SCBs of allocated spaces are queued by forward and backward pointers. There is one queue containing the SCBs of the real and static spaces, one that of the dynamic spaces and one that of the PMR spaces. The queue headers are located at begin of SCBATAB. After IPL, the SCBATAB and the queue headers are initialized as follows:

(n = NPARTS - number of static partitions(12))



ASCBFDYN is set, as long as there are dynamic partitions allocated.

In the following figure, two partitions have been allocated in class x, whereas the first partition X1 has already terminated. Space X2 is the currently active space. Note, that SCBPTR does never point to the SCB of the shared area.

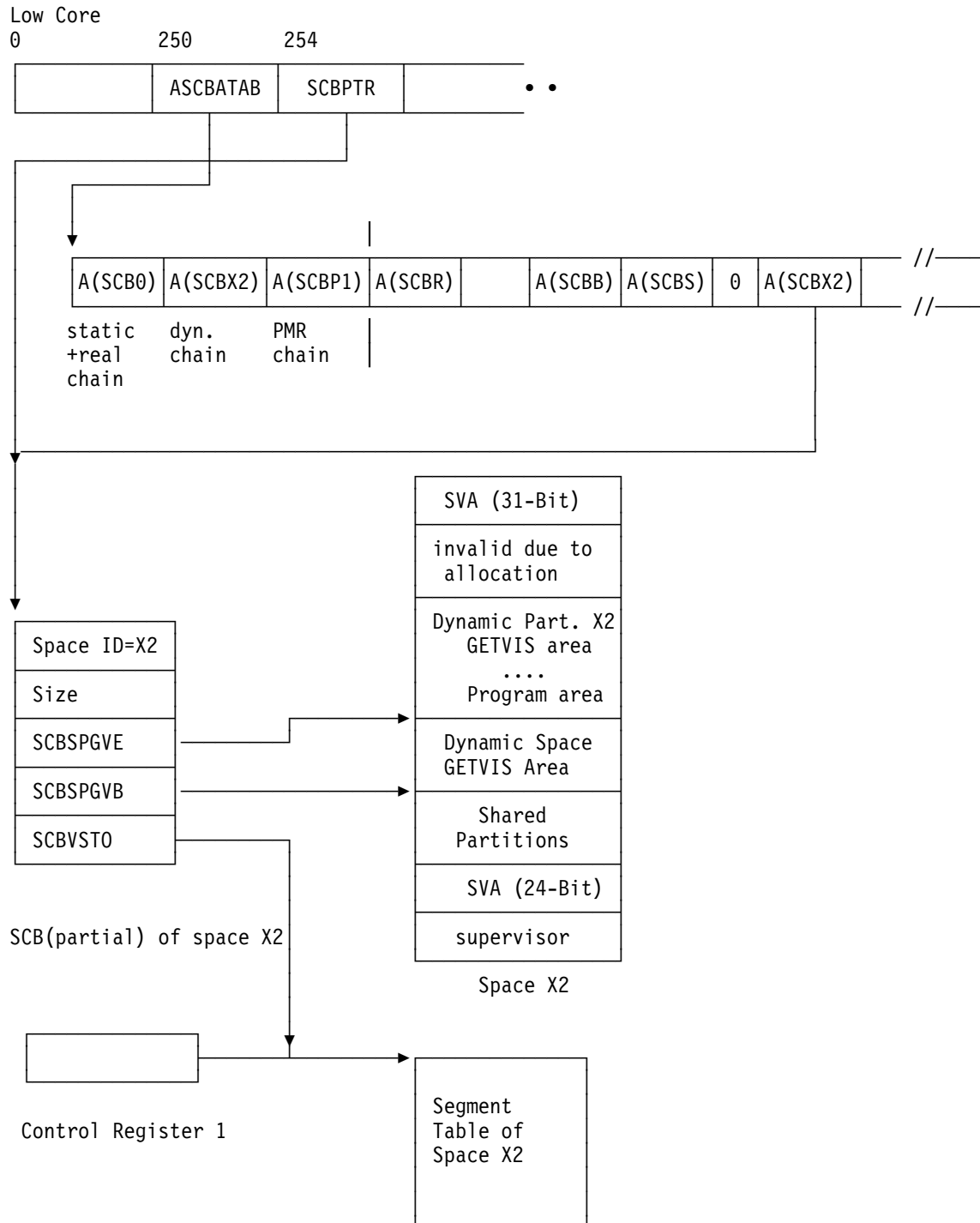


Figure 182. Space Control Block (SCB) Data Relationship

## Partition Control Blocks

The PCB, the PCE, the PCBX, the COMREG, the PIB and the PIB2 contain static and dynamic status information about the system and about partitions.

There exists one set of these control blocks for the system and for each partition.

The control blocks for the system and the static partitions are allocated during supervisor generation, the ones for dynamic partitions are dynamically allocated during partition allocation and freed during partition deallocation.

## Static Partition Control Blocks Interrelationship

This interface is only valid for static partitions and should no longer be used.

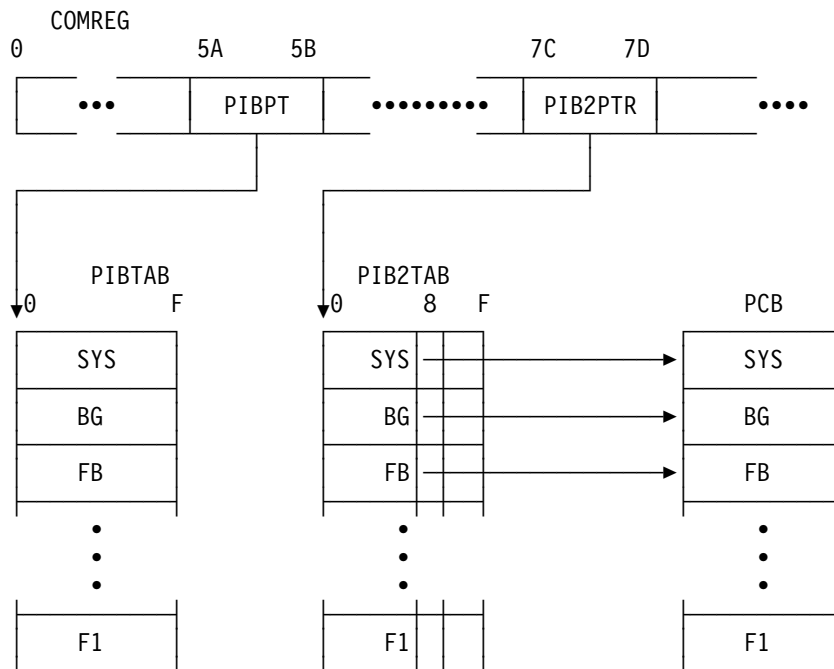


Figure 183. Partition Control Blocks Interrelationship (Static Partitions only)

The PIB/PIB2 for a given partition is found by adding the PIK of this partition to the begin address of the appropriate table.

The COMREG fields PIBPT and PIB2PTR and others (FICLPT, NICLPT, LUBPT, DIBPT and PDTABB) have a field length of 2 bytes only and are therefore not supported for dynamic partitions:



## Partition Control Blocks Interrelationship

The PCBATAB holds the PCB addresses of all generated static partitions and allocated dynamic partitions in 'PIK divided by 4' order. The APCBATAB field in low core gives access to the PCBATAB, which points to a dummy PCBATAB during IPL. This dummy PCBATAB contains only the PCB addresses for system and the static partitions.

The DYNCLASS ID=INIT service at the end of IPL allocates the PCBATAB for all possible partitions (defined by SYS NPARTS=..) in the system GETVIS area, initializes the table and updates the APCBATAB pointer. Not allocated dynamic partition entries of the PCBATAB are zero. X'FFFFFFFF' indicates the end of the PCBATAB.

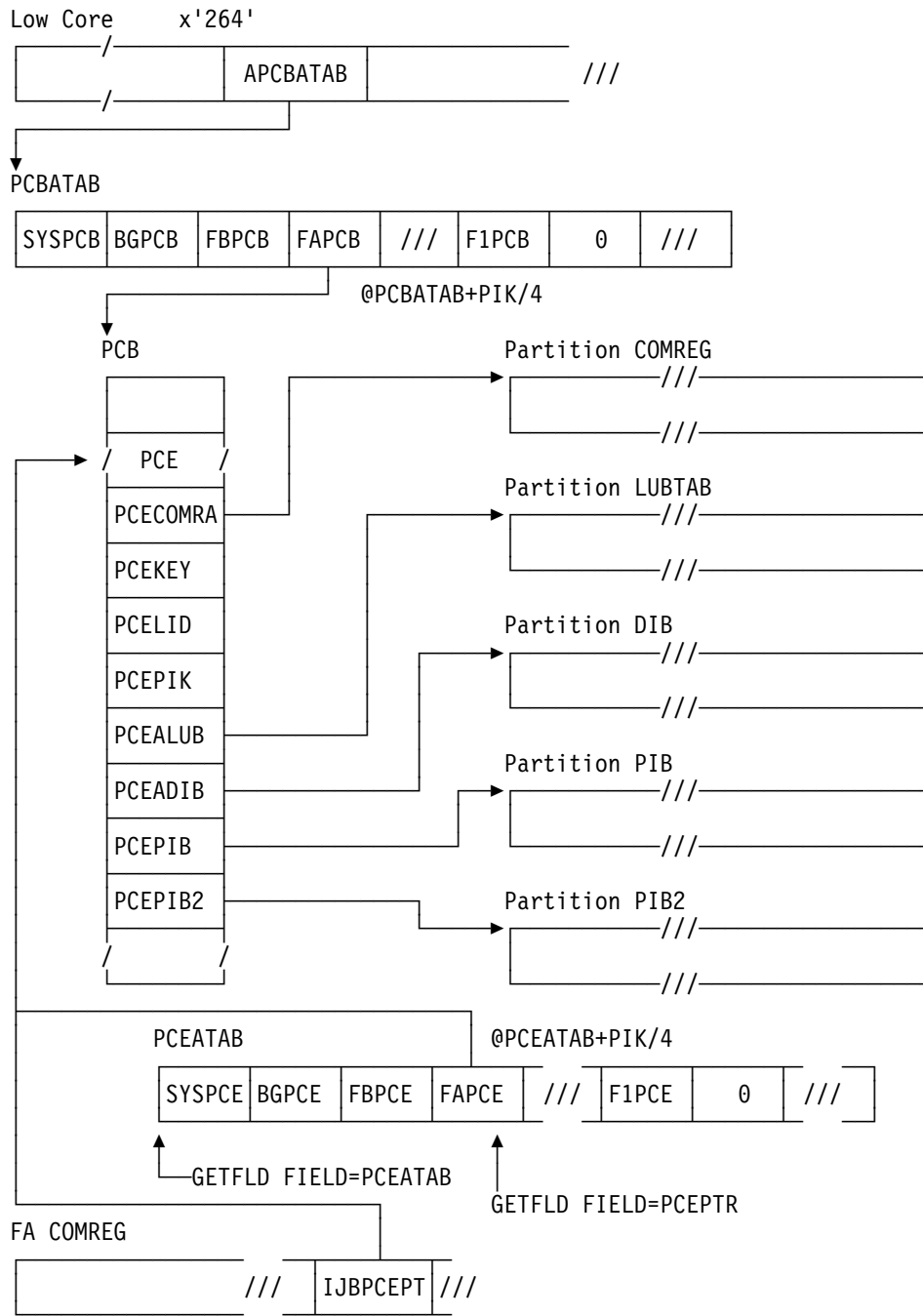


Figure 184. How to Address the PCB - PCE

---

## Task Control Blocks

The TIB, TCB and TCBX contain static and dynamic status information on system tasks and user tasks.

One set of these control blocks exists for each system task, for each main task and for each generated user subtask.

The control blocks for the system tasks and for the main tasks associated with static partitions are generated at supervisor generation time.

The control blocks for the tasks associated with dynamic partitions are (de-)allocated during dynamic partition (de-)allocation.

The control blocks for a user subtask are reserved during attach processing. They are not freed when the subtask is detached.

In order to minimize real storage requirements the TIBs, TCBs and TCBXs for subtasks and dynamic partitions are allocated in the SVA.

The TCBX may be located anywhere in storage whereas the TIB and TCB are located in the 24-bit area.

The TIBs are addressed via an address table (TIBATAB) with offset  $TID*4$ . The TIB contains all task-related information which has to be kept in fixed storage, either for logical or for performance reasons.

The TIBATAB is moved from the 24-bit area to the 31-bit area when support for more tasks is activated (via command `SYSDEF SYSTEM,NTASKS=nnn`).

The length of a TCB is dependant on the task for which it is generated and is contained in the TCB field `TCBLNGTH`.

The TCB field `TCBATCBE` contains a pointer to the TCBX.

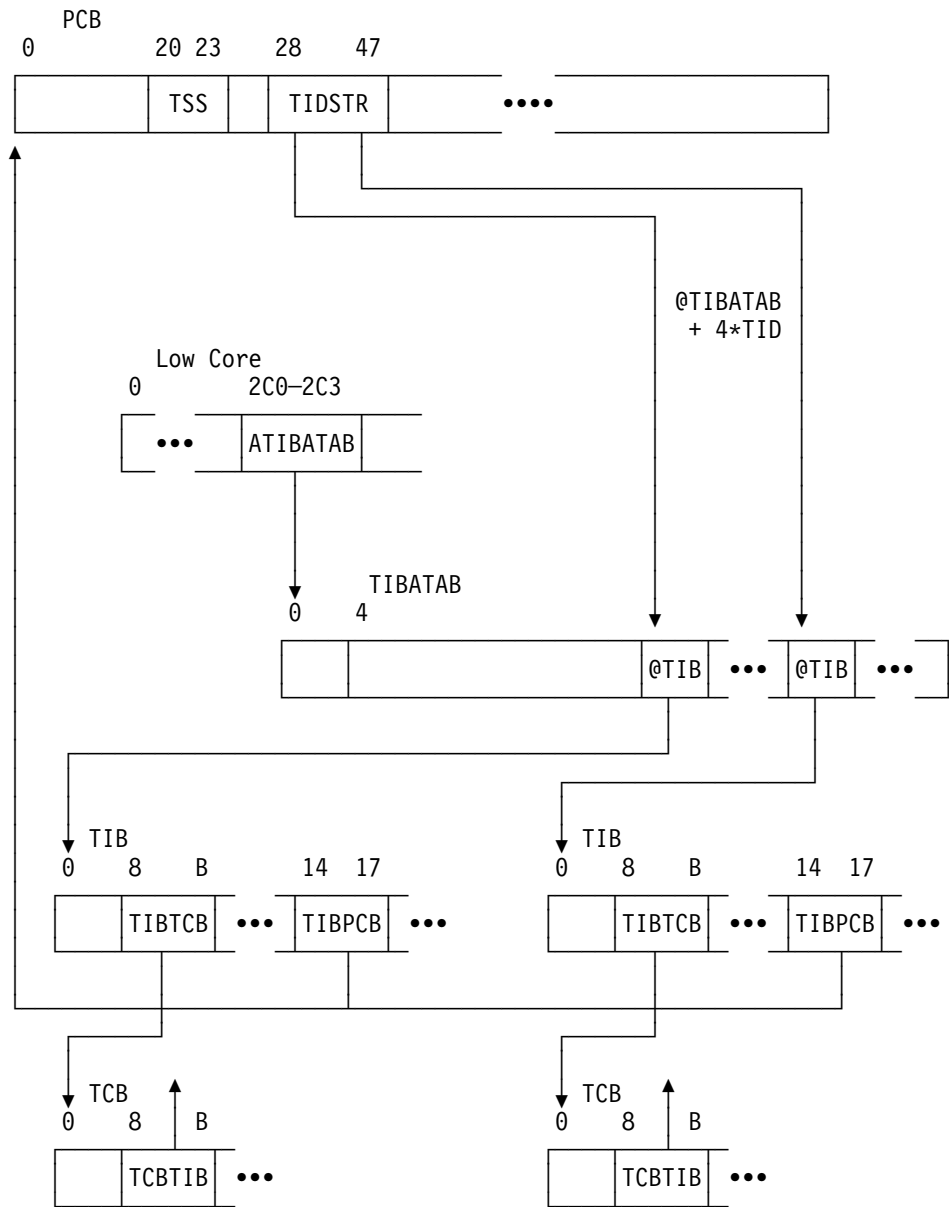


Figure 185. Partition/Task Control Table Relationship

## Save Areas

Problem Program (PP) Save Area  
 User Supplied Save Area (STXIT) described by macro MAPSAVAR  
 (refer to *z/VSE System Macros Reference*, SC33-8230).  
 LTA Save Area  
 System Save Area  
 Access Registers Save Area  
 Logical Transient Area Occupancy and Activity

The addresses of the various save areas allocated by the system can be found in the appropriate TCB table. The layout of the different Save Areas is shown in Figure 186 through Figure 189 on page 451.

### Problem Program (PP) Save Area

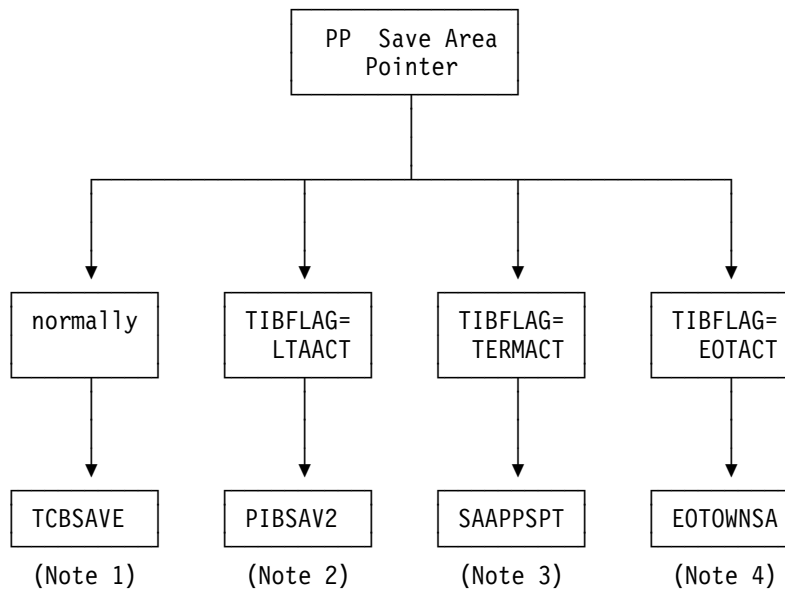
0 (0)	Program Name	7 (7)
Program Status Word (PSW) (Note 1)		
16 (10)	General Register save area (Reg. 9 through Reg. 8)	79 (4F)
80 (50)	Reserved 81 (51)   82 (52) (Note 2)	87 (57)
88 (58)	Floating Point Reg. save a. (Reg. 0 through Reg. 6)	119 (77)

#### Notes:

1. EC Mode PSW
2. Bytes 82 - 87
  - main task: Date of job begin
  - subtask: 82 (52) - 83 (53) : Reserved
  - 84 (54) - 85 (55) : Task id
  - 86 (56) : Key of ICCF pseudo-partition
  - 87 (57) : Reserved

Figure 186. Problem Program Save Area

## Where to Find the PP Save Area Pointer in Case of Termination



### Notes:

1. Located in Task Control Block (TCB).
2. Located in Partition Information Block (PIB).
3. Located in Dump Communication area (SAACOMM). Address to this area can be found in PCB at label PCBSAAPT.
4. Identified via "eye catcher" 'EOT SAVE' in the supervisor.

Figure 187. Problem Program (PP) Save Area Pointer in Case of Termination

## LTA Save Area

0 (0)	Logical Transient Phase name	7 (7)
Program Status Word (Note 1)		
16 (10)	General Register save area (Reg. 9 through Reg. 8)	79 (4F)
80 (50)	Reserved	87 (57)
88 (58)	Floating Point Reg. save a. (Reg. 0 through Reg. 6)	119 (77)

**Note:** EC Mode PSW

*Figure 188. LTA Save Area*

## System Save Area

Program Status Word (Note 1)		
8(8)	General Register save area (Reg. 9 through Reg. 8)	71 (47)

**Note:** EC Mode PSW

*Figure 189. System Save Area*

## Access Register Save Area

Each task has a first and second AR save area. The pointer to the first AR save area is contained in the TCB field TCBARSAV, the pointer to the second AR save area in the TCB field TCBAR2SV. The second AR save area is located within the TCBX. For system tasks and static partitions, the first AR save area is generated within the supervisor. For dynamic partitions, the first AR save area is (de-)allocated during dynamic partition (de-)allocation as it is done for all other control blocks, too. The the first AR save area for subtasks is allocated during ATTACH subtask processing.

## Logical Transient Area Occupancy and Activity

Indications of Logical Transient Area Occupancy and Activity					
Status	BGCOMREG	Attention PIB	Problem PIB	Condition of LTA	Notes
SVCs issued	Contents of LTK + 1 (1 Byte)	Address in ARFLG + 1 (3 Bytes)	Addr. in PIBSAVE+1 (3 Bytes)		
	zero	Logical Transient Save Area (LTASAVE)		Free	Initial condition before issuing SVC-02
SVC-02	Owner's Partition Identific. Key	Problem Program Save Area	Logical Transient Save Area (LTASAVE)	Active	
SVC-02 SVC-0B	zero	Logical Transient Save Area (LTASAVE)	Problem Program Save Area	Free	Restored to (1)
SVC-02 SVC-08	Owner's Partition Identific. Key	Logical Transient Save Area (LTASAVE)	Problem Program Save Area	Occupied but Inactive	SVC-08 may be issued only from LTA. General register 14 contains address of entry point to the user routine
SVC-02 SVC-08 SVC-09	Owner's Partition Identific. Key	Problem Program Save Area	Logical Transient Save Area (LTASAVE)	Active	Restored to (2). SVC-09 may be issued only from Problem Program.
SVC-02 SVC-08 SVC-09 SVC-0B	zero	Logical Transient Save Area (LTASAVE)		Free	Restored to (1)

Figure 190. Indications of Logical Transient Area Occupancy and Activity



# Control Blocks related to Lock Management

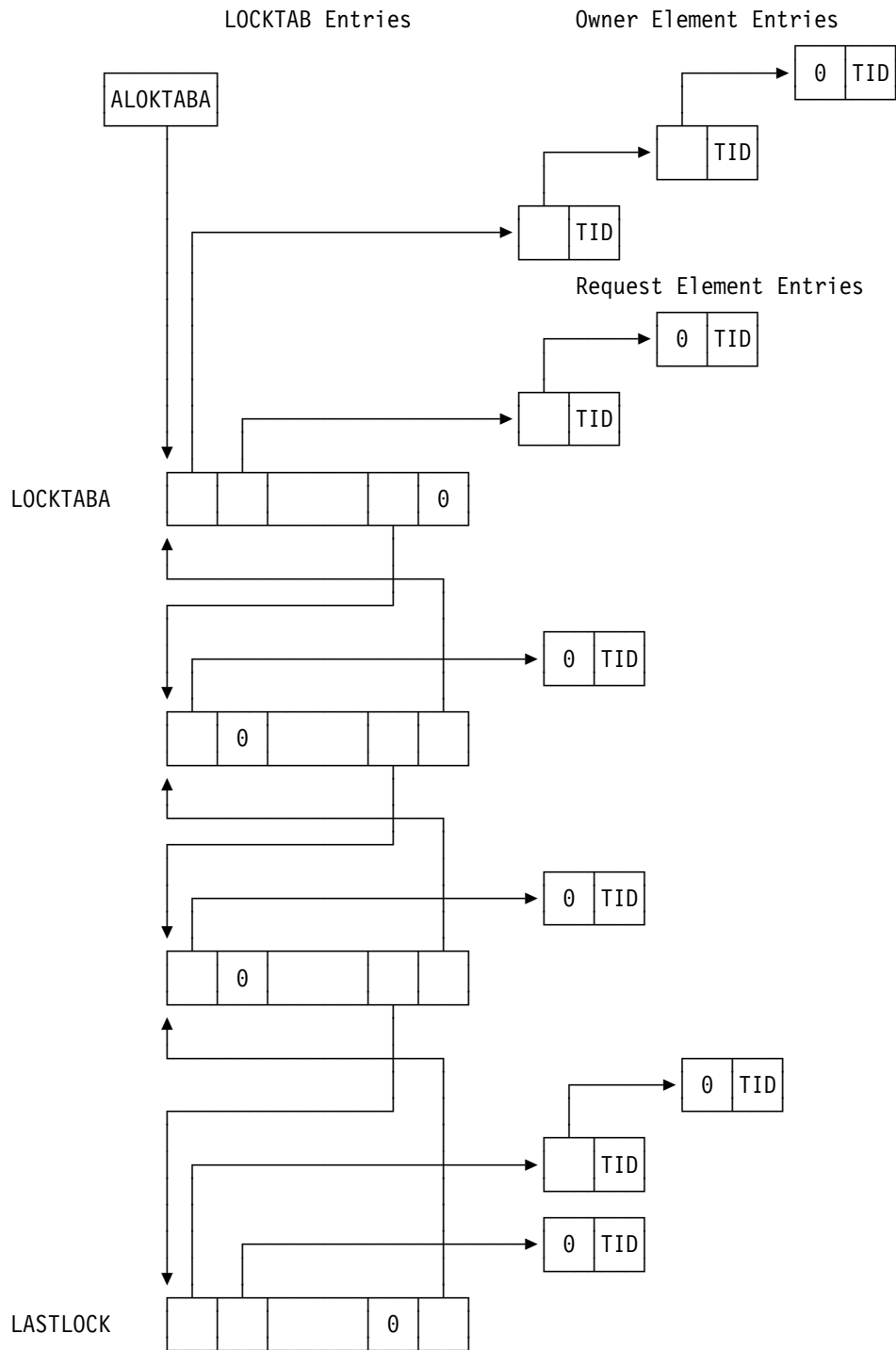


Figure 191. Relationship Between LOCKTAB, Owner Elements and Request Elements

## Event Control Block (ECB)

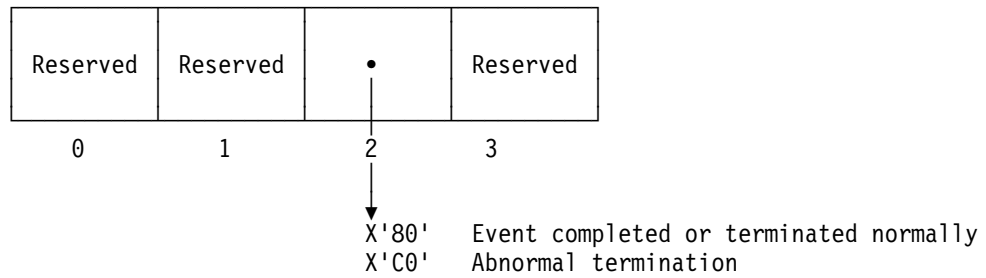


Figure 192. Event Control Block (ECB)

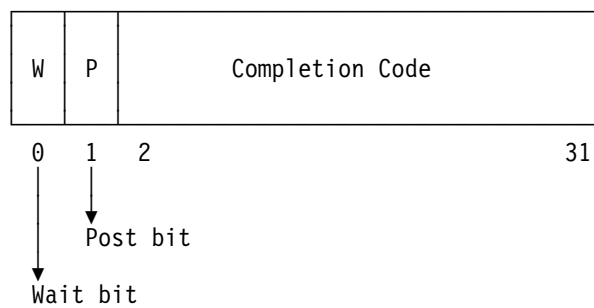


Figure 193. OS/390 Event Control Block (ECB)

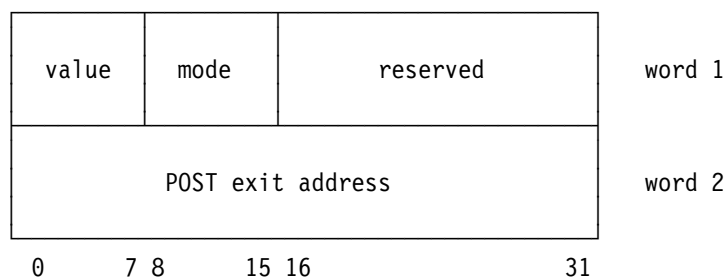
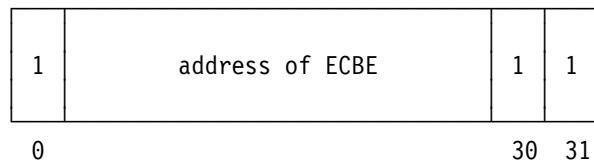
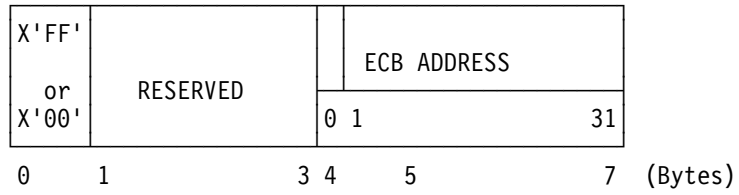


Figure 194. OS/390 Extended Event Control Block (ECBE)

## Resource Control Block (RCB)



Bytes	Description
0	X'FF' resource is in use X'00' resource is not in use
1 – 3	Reserved
4 – 7	bit 1-31: ECB address of current resource owner bit 0=1: another task waiting for this resource

Figure 195. Resource Control Block (RCB)

# VIO Control Blocks

VIO Communication Area (VIOCM)  
 VIO Table Entry (VTABE)

VIO File Identification Entry (VIOTABE)  
 VIO Block Table Entry (BLKTBE)

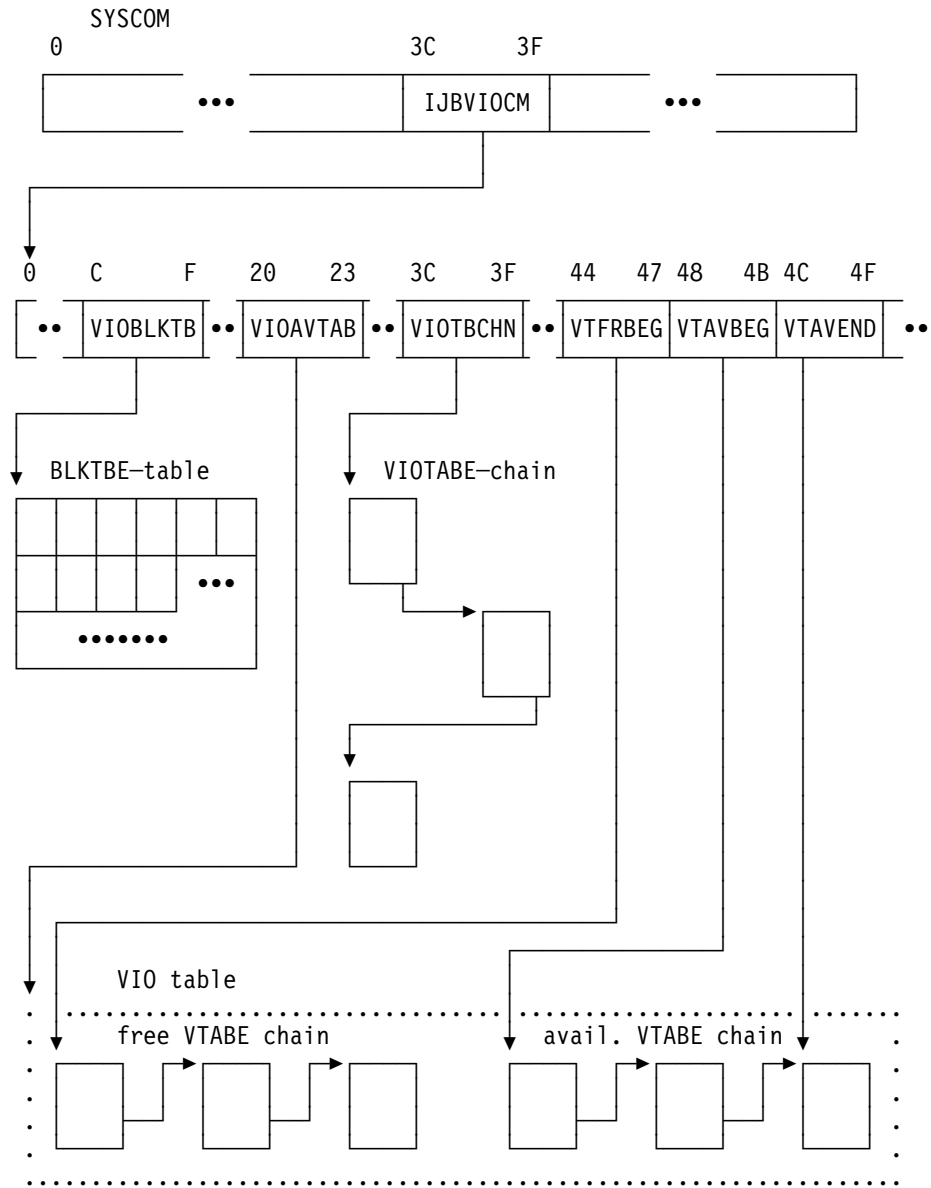


Figure 196. VIO Control Block Relationship (after IPL)

## VIO Communication Area (VIOCM)

Bytes		Label	Description
Dec	Hex		
..... Temporary space used during IPL .....			
0-3	0-3	VIOARBEG	Begin of VIO tables (set by IPL)
4-5	4-5	VIOSGM#	Number of VIO segments (set by IPL)
6-7	6-7	VIOSPSIZ	Number of bytes allocated per VIO segment
8-9	8-9	VIOVPSIZ	Number of bytes to be allocated per page in VPOOL
10-11	A-B	VIOKSGSH	Shift value: K-bytes and segment no.
..... Normal layout of VIOCM after IPL .....			
0-3	0-3	VIOSPBEG	Begin of VIO allocation string
4-7	4-7	VIOSPEND	End of VIO allocation string
8-11	8-B	VIOSPNXT	Next segment slot to check
12-15	C-F	VIOBLKTB	Address of VIO Block Table
16-19	10-13	VIOVPPSZ	Size of VPOOL in pages
20-23	14-17	VIOVPSZE	Size of VPOOL in bytes
24-27	18-1B	VIORECN1	Record number of 1st VIO block on PDS
28-31	1C-1F	VIOVPEPA	EPA of 1st VPOOL page
32-35	20-23	VIOAVTAB	Address of VTAB
36-39	24-27	VIOBLKSZ	Size of a VIO block
40-43	28-2B	VIOSEGSZ	Size of a VIO segment
44-45	2C-2D	VIOBLKOS	shift amount to get offset in BLKTAB out of block number
46	2E	VIOSWTCH	Flag byte
		VIOERR25	X'80' don't set 'status saved' flag
47	2F		Reserved
48	30	VIOBLDSC	OR-byte for disconnected page/frame
49-51	31-33		Reserved
52-59	34-3B	VIOPLID	VIO GETVIS subpool ID
60-63	3C-3F	VIOBTCHN	VIOTAB chain header
64-67	40-43	VIOOPCNT	Number of VIOTAB entries
..... Header for queue of free VTAB entries .....			
68-71	44-47	VTFRBEG	Address of first element in chain
..... Header for queue of available VTAB entries .....			
72-75	48-4B	VTAVBEG	Address of first element in chain
76-79	4C-4F	VTAVEND	Address of last element in chain
.....			
80-83	50-53	AVIOFBLK	Entry address of VIOFRBLK routine
84-87	54-57	AVIOFPAG	Entry address of VIOFRPAG routine
88	58		Length of VIO communication area

Figure 197. VIO Communication Area (VIOCM)

## VIO Table Entry (VTABE)

Bytes		Label	Description
Dec	Hex		
0	0	VTFLG	Flag byte
		VTPAGERR	X'80' Page in error
1	1	VTUSCNT	Usage count
2-3	2-3	VTOWNER	Partition ID of requester
4-6	4-6	VTFRAM	Frame number belonging to page
7	7	VTPFCNT	Number of pending page-faults
8-11	8-B	VTBLKN	Total block number of page
12-15	C-F	VTFPTR	Forward pointer
16	10	LVTABE	Length of VTABE

Figure 198. VIO Table Entry (VTABE)

## VIO File Identification Entry (VIOTABE)

Bytes		Label	Description
Dec	Hex		
0-1	0-1		Reserved
2	2	VIORBCM1	Communication byte
		VIORBTRB	X'80' VIO POINT request complete
3	3	VIORBRTC	Return code
		VIORBEOF	X'04' Requested block outside area
		VIORBERR	X'08' Unrecoverable error
		VIORBINC	X'0C' Inconsistent state
4-7	4-7	VIORBASZ	Actual size of area in bytes
8-11	8-B	VIORBBSZ	Size of a block in bytes
12-15	C-F	VIORBPNT	Virtual address of current block = 0 : No VIO POINT given up to now < 0 : VIO POINT in process
16-19	10-13	VIORBRBA	Relative byte addr. of current block
20-23	14-17	VIORBASR	Address of service routine
24-31	18-1F	VIOTBSID	Storage ID for validation
.....		VIOTIB .....	Pseudo-TIB for VIO .....
32-43	20-2B		1st three fullwords of TIB
44	2C		not used in VIO TIB
45	2D	VIOTIBFL	TIBFLAG1 in VIOTIB
		VIOIND	X'A0' Indication for VIO TIB
46-47	2E-2F		not used in VIO TIB
48-51	30-33		Next fullword of TIB
52-53	34-35	VIORTID	TID of VIO POINT requester
54-55	36-37	VIOOWNER	PIK of owner partition
56-59	38-3B	VIOPCB	PCB addr of VIO-owner
60-63	3C-3F		not used in VIO TIB
64-67	40-43	VIOPFSCB	SCB where page I/O req. belongs to
68-71	44-47		not used in VIO TIB
.....		..... End of	Pseudo-TIB for VIO .....
72-75	48-4B	AFLSEGTB	Address of 1st file segment block
76-79	4C-4F	VTABEACT	Address of VTABE belong. to VIORBPNT
80	50	VIORQKEY	Storage Key of Requestor
81-95	51-5F		reserved
96-99	60-63	VIOBLKN	Save area for total block number
100-101	64-65	VIOTBOPT	Option bytes from VIOPL
100	64	VIOTBLFT	Scope option from VIOPL
101	65	VIOFLAG	Flag byte
		ASYNCH	X'80' Asynchronous request
		VIOTSAV	X'40' Status already saved
102-103	66-67		Reserved
104-107	68-6B	VIOTBNXT	VIOTABE chain pointer
107	6B	VIOTABLN	Length of VIOTABE - 1

Figure 199. VIO File Identification Entry (VIOTABE)

## VIO Block Table Entry (BLKTBE)

The VIO Block Table Entry layout corresponds to the layout of the Page Table Entry.

Bytes		Label	Description
Dec	Hex		
0-7	0-7	BLKPAG64	Page / frame addr. belonging to block
0	0		Filler
5	5	BLKKEY	Storage key
6	6	BLKSTAT	Status indication
		BLKFRC0	X'06' Frame connected to block if both bits off, page connected to block (PGCON)
		BLKDISC	X'04' Whether page nor frame conn.
7	7	BLKERR	X'02' Error on block
		BLKSTAT2	second status byte
		BLKPDS	X'01' Copy on external storage
8	8	LBLKTBE	Length of block table entry

Figure 200. VIO Block Table Entry (BLKTBE)



---

## OS/390 Control Blocks

To ship object code compatible to OS/390 macros, VSE supports the OS/390 system function table, which is anchored in the OS/390 communication vector table (CVT) at label CVTXSFT. The address of the CVT is located in low core at location X'10'. In VSE the CVT and system function table are allocated within the supervisor.

**Note:** The z/VSE (operating system) identificaton in the CVTDCB field is unique compared to the operating systems z/VM and z/OS. .

**Note:** In VSE/ESA 1.1.0 and earlier releases the X'10' low storage location is used for the VTAM vector table (AT CVT) address. Beginning with VTAM 3.4 the VTAM vector table address can be obtained from X'408' (same location as in MVS). Because of transparency reasons the new location is freed up. that is the VTAM vector table address is located in X'10' as well as X'408'. Beginning with VSE/ESA 1.3.0 only X'408' holds the VTAM vector table address and X'10' the MVS CVT address.

Beginning with VSE/ESA 2.4.0, VSE supports an OS390 emulation mode. When a partition runs in OS390 emulation mode, part of the OS/390 control block structure is built.

z/VSE supports those control blocks(fields) that are either used by CICS or are required by the OS/390 API macros.

The layout of control blocks that were ported completely to VSE is identical with the OS/390 layout. But only required fields are set by VSE.

All control blocks are located below 16 MB.

The control block handling is done in phase IJBMCBE.

## Allocation

OS/390 control blocks are allocated when a partition runs in OS/390 emulation mode, EXEC ...,OS390. Emulation mode is only allowed when default allocation is used, i.e. shared and real partitions are not supported.

For the maintask, address space and task related control blocks are allocated during SVC 133 processing.

For the subtasks, task related control blocks are allocated during ATTACH (VSE) and ATTACHX(OS/390) processing, i.e in an emulated partition, OS/390 control blocks exist even for subtasks attached by VSE ATTACH.

To allocate OS/390 control blocks, phase IJBMCBE is called by SGAP using the interface macro

SMCBEM FUNC=ALLOC.

## Deallocation

Maintask control blocks are deallocated during maintask termination called by EOT processing.

Subtask control blocks are deallocated during subtask termination called by DETACH processing.

To deallocate OS/390 control blocks, phase IJBMCBE is called by SGAP using the interface macro

SMCBEM FUNC=DEALL.

## OS/390 TCB Structure

The job step task (tcb) is the maintask (tcb) in VSE. A tcb is anchored to the attaching task tcb. At task termination, all underlying tcbs are detached by the system.

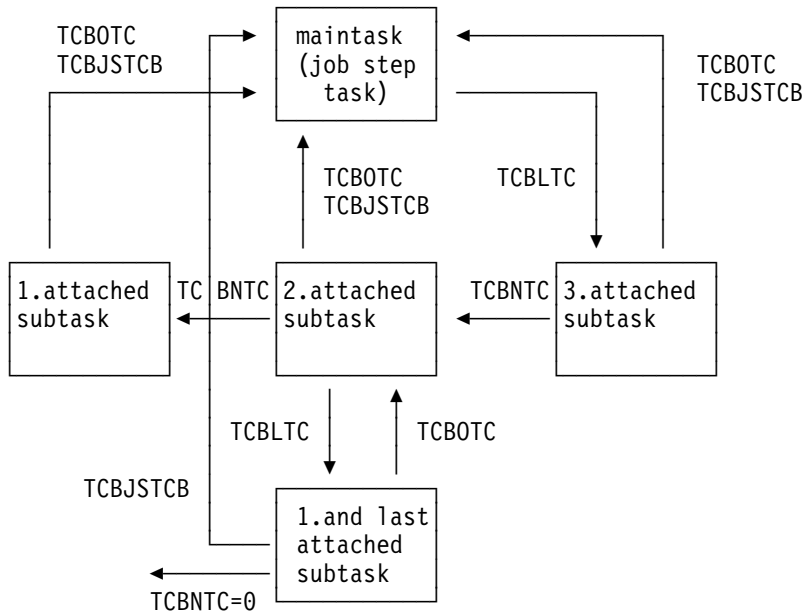


Figure 201. OS/390 TCB Structure

- TCBOTC: points to TCB of attaching task
- TCBLTC: points to TCB of last attached task
- TCBNTC: points to TCB of previous attached task
- TCBJSTCB: points to TCB of maintask (job step)

Starting with a task's TCBLTC, one can find all tasks that were attached by this task by TCBNTC.

## RB Structure in VSE

Request Blocks are used to save the status when an interrupt occurs.

VSE does not have the RB structure OS/390 has.

The RB emulation in VSE is done the following way:

- VSE supports RB types PRB and SVRB
- each task has a TCB and PRB
- whenever a task issues the CICS SVC a SVRB (and XSB) is allocated
  - the SVRB is only allocated for the CICS SVC
  - during CICS SVC processing, the supervisor calls the function `SMCBEM FUNC=UPDATE` to enqueue a SVRB in the RB chain.
  - on return from the CICS SVC, the supervisor calls the function `SMCBEM FUNC=UPDATER` to remove the SVRB from the RB chain.
  - a nesting level of 4 CICS SVCs is supported
- when the CICS SVC routine gets control, the SVRB is enqueued and all fields required by CICS are set.

## RB Structure After Task Initialization

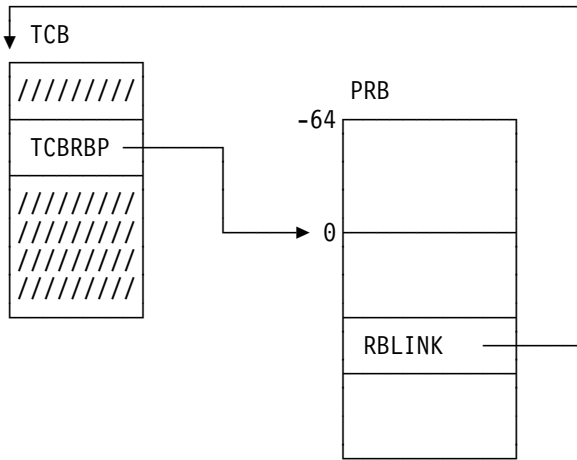


Figure 202. Request Block Structure After Task Initialization

### PRB Contents

- RBLINK points to TCB
- RBSTAB1 set to X'00' (Indicates this is a PRB)
- RBSTAB2 set to X'80' (indicates this RB points to a TCB)

## RB Structure After First CICS SVC

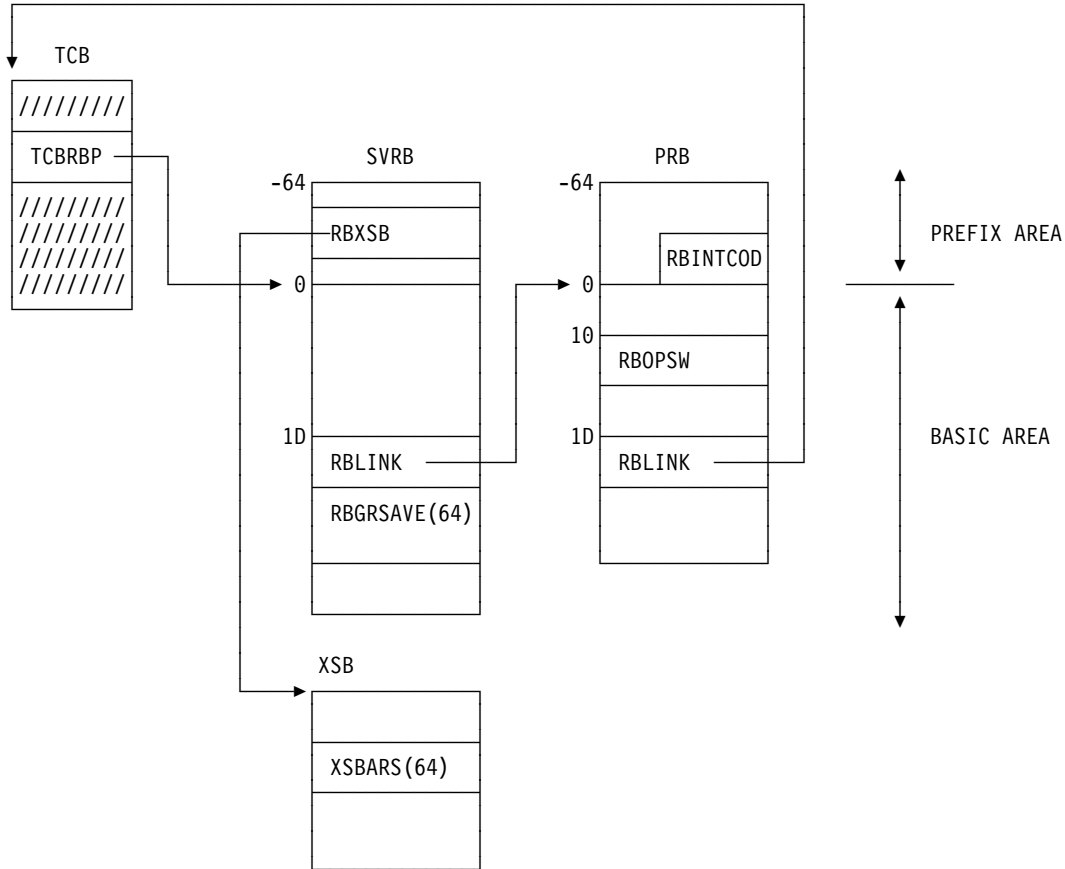


Figure 203. RB structure on entry to first CICS SVC processing

### SVRB Contents

- RBLINK points to first PRB basic section
- RBSTAB1 set to x'C0' (indicates this is an SVRB)
- RBGRSAVE (RBGRS0-RBGRS15) contains registers at the time of the (first) CICS SVC interrupt
- Allocated space include RBEXSAVE area
- RBFEPARM area allocated and supported as part of FESTAE processing

### PRB Contents/XSB contents

- RBLINK points to TCB
- RBSTAB1 set to X'00' (Indicates this is PRB)
- RBSTAB2 set to X'80' (indicates this points to TCB)
- PRB prefix contains interrupt code (RBINTCOD=CICS SVC no=150)
- RBOPSW contains PSW at time of (first) CICS SVC
- XSBARS contains access registers at the time of the (first) CICS SVC interrupt (not used by CICS).

## RB Structure After Second CICS SVC

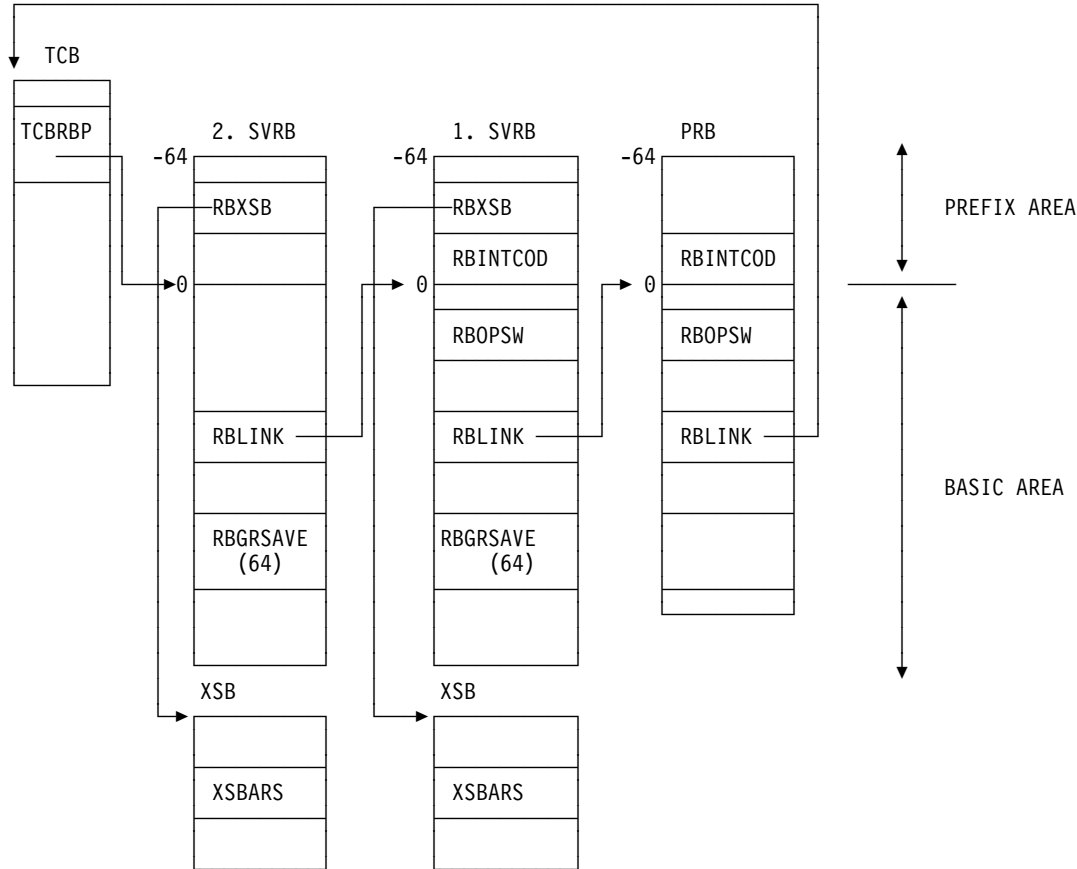


Figure 204. RB structure on entry to second CICS SVC processing

### PRB Contents

Unchanged

### SVRB Contents Of First SVRB

- RBLINK points to PRB basic section
- RBSTAB1 set to x'C0' (indicates this is an SVRB)
- RBOPSW contains PSW at time of (second) CICS SVC
- RBGRSAVE (RBGRS0-RBGRS15) contains registers at the time of the (first) CICS SVC interrupt
- RB prefix contains interrupt code (RBINTCODE = CICS SVC number = 150)
- RBXSB pointer to XSB

### SVRB Contents Of Second SVRB

- RBLINK points to first SVRB basic section
- RBSTAB1 set to x'C0' (indicates this is an SVRB)
- RBGRSAVE (RBGRS0-RBGRS15) contains registers at the time of the (second) CICS SVC interrupt
- Allocated space include RBEXSAVE area
- RBFEPARM area allocated and supported as part of FESTAE processing
- RBXSB pointer to XSB

## **Anchor of OS/390 Control Blocks in VSE**

- TCBX.TCBXMVST: addr of OS/390 TCB
- SCB.SCBASCB: addr of OS/390 ASCB
- OS/390 TCB.TCBVSETC: addr of VSE TCB

---

## Appendix B. I/O Control Blocks

---

### Basic Input/Output Control Words (z/Architecture)

During Input/Output processing on z/Architecture hardware the following control blocks are used:

- CCW
- ORB (Operation-Request Block)
- IRB (Interruption-Response Block)
- SCHIB (Subchannel Information Block)

For a detailed description of the control blocks refer to the appropriate mapping Macros and/or to *z/Architecture Principles of Operation, SA22-7832*.

For compatibility reasons with S/370 architecture the following control words are also used:

- CAW
- CSW

Figure 205 to Figure 206 on page 468 show the layout of the Channel Address Word (CAW) and the Channel Status Word (CSW). For more information refer to the appropriate *Principles of Operation* manual.

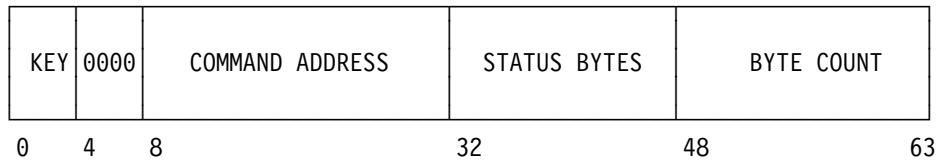
### Layout of CAW



Bits	Description
0 – 3	Storage protection key
4 – 7	Reserved (must be zero)
8 –31	Address of first/only CCW

Figure 205. Channel Address Word (CAW)

## Layout of CSW



Bits	Apprev.	Description
0 – 3		Storage protection key
4		Reserved (must be zero)
5		Logout pending
6 – 7		Deferred condition code
8 – 31		Address+8 of last CCW executed
32	ATTN	Attention
33	SM	Status modifier
34	CUE	Control unit end
35	BSY	Busy
36	CE	Channel end
37	DE	Device end
38	UC	Unit check
39	UX	Unit exception
40	PCI	Program controlled interruption
41	IL	Incorrect length
42		Channel program check
43		Channel protection check
44		Channel data check
45		Channel control check
46		Interface control check
47		Channel chaining check
48 – 63		Residual byte count

Figure 206. Channel Status Word (CSW)



# Input/Output Control Blocks and Areas

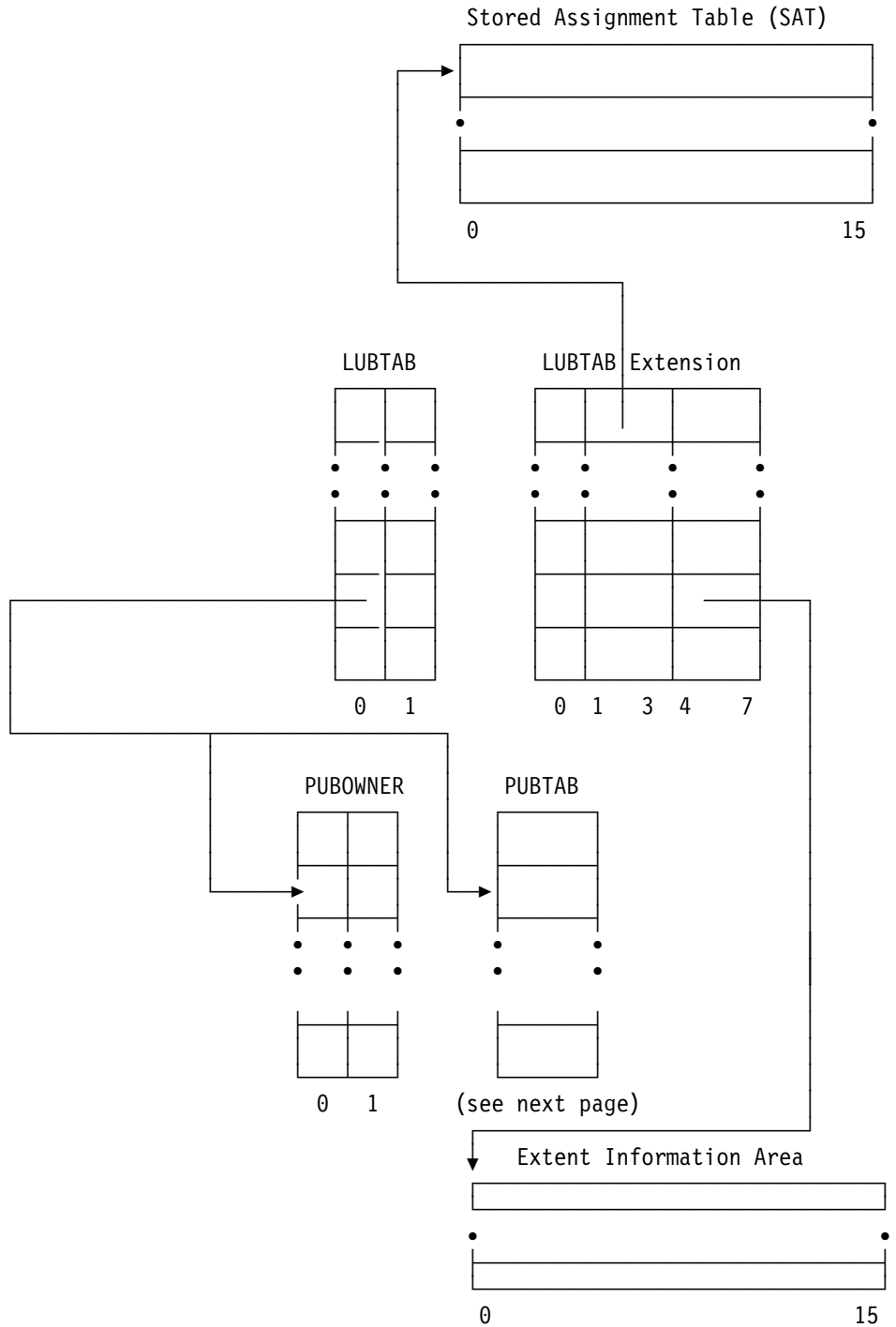


Figure 207 (Part 1 of 4). I/O Table Interrelationship

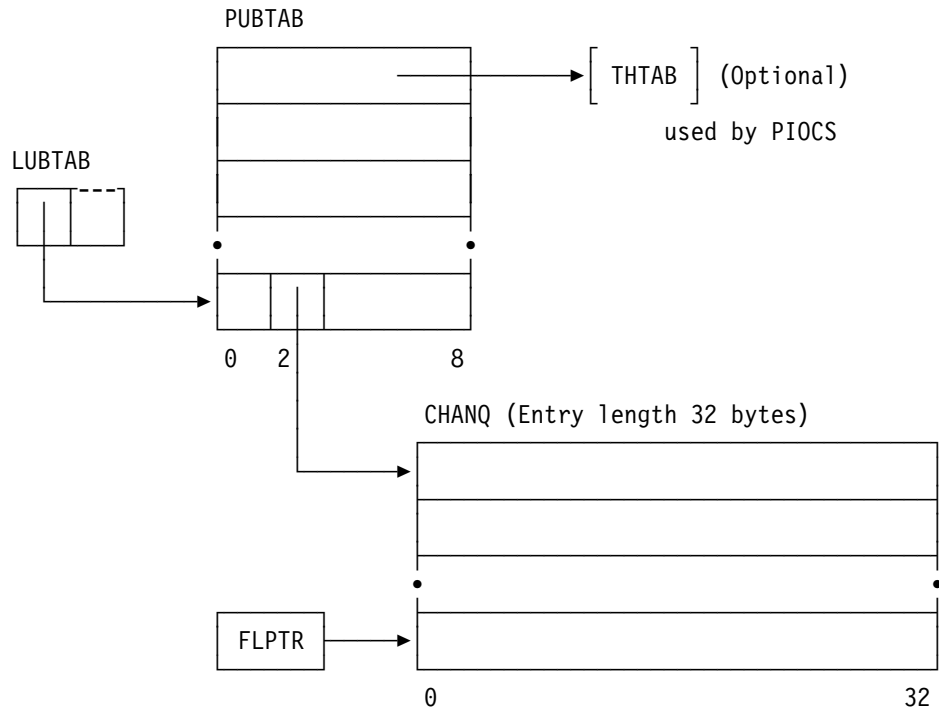


Figure 207 (Part 2 of 4). I/O Table Interrelationship

Label	Description
LUBTAB  (Logical Unit Block Table)	Byte 0 of each entry is the low order byte of an index pointer. Byte 1 of each entry is the high order byte of an index pointer (or X'FF' if IODEV->254). Is index in the PUB Table (PUBTAB) and PUB Ext. Area (PUBXAREA) as well as to the PUB OWNERSHIP Table (PUBOWNER). X'FFFF' indicates that no logical unit is assigned. X'FEFF' indicates that I/O requests are to be ignored.
LUBTAB EXTENSION	Bytes 1-3 point to first STORED ASSIGNMENT TABLE entry. Bytes 4-7 point to first EXTENT INFORMATION AREA entry. Zero indicates no extent information available.
EXTENT INFORMATION	Bytes 1 - 3 point to the next EXTENT INFORMATION ENTRY. Zero identifies this entry as the last one in the chain.
STORED ASSIGNMENT ENTRY	Bytes 1 - 3 point to the next STORED ASSIGNMENT TABLE entry.
PUBTAB  (Physical Unit Block Table)	Byte 2 is the LOW index byte into the CHANQ Table X'FF' indicates that no request is queued to the PUB. Byte 3 is the HIGH index byte into the CHANQ (IODEV>254) Byte 5 is an index pointer which for: DASD points to the entry in the TRACK HOLD TABLE (THTAB)
PUBXAREA  (Physical Unit Block Extension Area)	Bytes 0 - 3 contain the address of the associated PUBX entry.
PUBX  (Physical Unit Block Extension Entry)	Device specific information used internally.

Figure 207 (Part 3 of 4). I/O Table Interrelationship

Key	Description
FLPTR (Free List Pointer)	This one-byte pointer contains the entry index of the next free entry in the Channel Queue Table (CHANQ). X'FF' in this field indicates: No more free CHANQ entry More device (IODEV>254) supervisor was IPL'ed
CHQFLPTR (Free List Pointer)	This two-byte pointer contains the entry index of the next free entry in the Channel Queue Table (CHANQ).
CHANQ (Channel Queue Table)	Byte 1 to 3 of this field contain the address of the begin of the Channel Queue Table (CHANQ).
THTAB (Track Hold Table)	Byte 0 on each entry points to the next entry in the chain of requests for a track/block to be held on a specific DASD (or the next free entry if in the free list) or it contains X'FF' if the entry is the last in a chain. Byte 12 contains a backward pointer. The backward pointer of the first Track Hold Table entry contains the PUB index.

Figure 207 (Part 4 of 4). I/O Table Interrelationship

## Logical Unit Block Tables (LUBTAB, LUBX, SAT, Ext.Inf.)

Logical Unit Block Table (LUBTAB)  
LUBTAB Extension Table  
Stored Assignment Table Entry (SAT)  
Extent Information Entry

### Logical Unit Block Table (LUBTAB)

The LUB tables for the static partitions are allocated within the supervisor. Label LUBTAB identifies the first byte of the tables, which begins with the BG system LUBs, followed by the BG programmer LUBs and the LUB pool for the static partitions F1,...,FB.

The Lub tables for the dynamic partitions are allocated during dynamic partition allocation.

Label PCEALUB of PCE contains the address of the partition's LUB table.

Logical Unit Block Entry (Note 1):

Bytes		Description
Dec	Hex	
0	0	Low order byte of PUB index of device assigned to this logical unit or X'FF' if no PUB is assigned or X'FE' if I/O is to be ignored for this log. unit
1	1	High order byte of PUB index of device assigned to this logical unit or X'FF' if IODEV<255, if no PUB is assigned or if I/O is to be ignored for this log. unit

Figure 208 (Part 1 of 2). Logical Unit Block (LUB) Entry and TABLE

Logical Unit Block Table:

(Note 2)								
SYSRDR	0		SYSRLB	10		SYSLIB	1E	
SYSIPT	2		SYSRLB	12		(Note 3)		
SYSPCH	4		SYSUSE	14			•	•
SYSLST	6		SYSREC	16		(Note 4)	•	•
SYSLOG	8		SYSCLB	18		SYS000		
SYSLNK	A		SYSDMP	1A		SYS001		
SYSRES	C		SYSCAT	1C		•	•	•
						•		
						SYSnnn		

**Notes:**

1. Null entries X'FFFF' are generated at supervisor generation time or during dynamic partition allocation.
2. There are 14 externally known system LUBs and one internally used for label access method.
3. System LUBs used by dynamic assignments.
4. The total number of system LUBs is a constant.

Figure 208 (Part 2 of 2). Logical Unit Block (LUB) Entry and TABLE

## LUBTAB Extension Table

The LUB Extension Table for each static partition is allocated and initialized by IPL. It has as many entries as allocated to the LUB table of that Partition. Each entry is 4 bytes long except the user did specify DASDFP=YES (IPL SYS-command) in which case each entry is 8 bytes in length.

The LUB Extension Table for a dynamic partition is allocated during dynamic partition allocation.

The start address of the LUB Extension Table is stored at label LUBEXT of the Partition Communication Region.

Bytes		Label	Description
Dec	Hex		
0	0	LUBXFLG LUBXPA  LUBXTA  LUBXPE	Flag Byte X'80' Permanent alternate assignment stored 40 Temporary alternate assignment stored 20 Permanent assignment stored 10 Reserved 08 Reserved 04 Reserved 02 Reserved 01 Reserved
1-3	1-3	LUBXSPT	(LUBXPA and/or LUBXTA is on) Pointer to first Stored Assignment Table entry (SAT)
1 2-3	1 2-3	LUBXPER	(LUBXPA and LUBXTA both off) Reserved Stored permanent assignment
OPTIONAL DASDFP=YES 4-7	4-7	LUBXEPT	Pointer to first EXTENT INFORMATION chain entry or zero if no EXTENT INFORMATION available

Figure 209. Logical Unit Block (LUB) Extension Entry

## Stored Assignment Table Entry (SAT)

The LUB Extension table entry may contain a pointer to a chain of assign entries, each containing additional information on stored assignments. Each entry is fixed length and is allocated in the System GETVIS area.

Bytes		Label	Description
Dec	Hex		
0	0	SATFLG	Flag byte
			X'80' Reserved
			40 Reserved
		SATPE	20 Permanent Assignment saved in this entry
			10 Reserved
			08 Reserved
			04 Reserved
			02 Reserved
			01 Reserved
1-3	1-3	SATNEXT	Pointer to next assign entry in the chain
4	4	SATEOCH	Offset within SATSAV of next free entry
5	5	SATEOPCH	Offset within SATSAV of saved permanent assignment
6-7	6-7	SATSAV	Space for saving permanent assignment (max. of 5)
...	...		
14-15	E-F		

Figure 210. Stored Assignment Table Entry (SAT)



## Extent Information Entry

The LUB extension table entry contains a pointer to a chain of Extent entries for DASD File Protection. Each entry is fixed length and is allocated in the System GETVIS area.

Bytes		Label	Description
Dec	Hex		
0	0	EXBFLG EXBREAD  EXBSHORT	Flag Byte X'80' Allow READ access only (no multi-track operation) 40 Extent information is CC only 20 Reserved 10 Reserved 08 Reserved 04 Reserved 02 Reserved 01 Reserved
1-3	1-3	EXBNXT	Pointer to next Extent entry in the chain or zero if this is the last Extent entry
4-7	4-7	EXBHI	High Extent Limit CKD Device      Cylinder+Head No. FBA Device      Physical Block No.
8-11	8-B	EXBLOW	Low Extent Limit CKD Device      Cylinder+Head No. FBA Device      Physical Block No.
12-13 14-15	C-D E-F	EXBCOUNT	Usage count for this extent Reserved

Figure 211. Extent Information Entry

## Physical Unit Block Tables (PUBTAB, PUBX, PUB2, PUBOWNER)

Physical Unit Block Table (PUBTAB)  
Physical Unit Block Extension (PUBX)  
Physical Unit Block 2 (PUB2)  
PUB Ownership Table (PUBOWNER)

### Physical Unit Block Table (PUBTAB)

Bytes 64-65 (X'40'-X'41') of the Partition Communication Region contain the address of the PUB table. Label PUBTAB identifies the first byte of the table.

Bytes		Label	Description
Dec	Hex		
0	0	PUBCHANN	Channel number of device (Hex 0-F) X'FF' indicates end of PUBTAB
1	1	PUBDEVNO	Unit number
2	2	PUBCHQPT	Index to first CHANQ entry X'FF' indicates no request enqueued
3	3	PUBCHQPH	High Order CHANQ index or 0
4	4	PUBDEVTY	Device type code
5	5	PUBOPTN	For TAPE devices: Tape Mode from ADD or ASSGN For DASD-Devices: Index of TRKHLD Table entry or X6 For 3704/3705: Type of channel adapter For 3800: Bit 0-1 00 3800 01 3800 B 10 3800 C 11 3800 BC
6	6	PUBOPTVD PUBOPDMY PUBCSFLG DEVBSY	C'V' Virtual FBA (not operational) X'80' Dummy console device Channel Scheduler flags X'80' Device is active 40 Reserved 20 Reserved
		QEDERR OPINTV INTPEND BRSDEV	10 I/O error queued for recovery 08 Operator intervention required 04 Interrupt was trapped by SDAID 02 Burst or overrunable device 01 Reserved
7	7	PUBJCFLG	Job Control flags Bits 0-4: TAPE : Standard MODE assignment
		PUBDVCUP	Not TAPE : All ones if device is up Device DOWN: All zeros
		RPSDASD PUBPPD	5: Device supports RPS 6: Device is not operational
Note: A PUB entry must be added during IPL for any device of the installation			

Figure 212. Physical Unit Block (PUB) Entry

## Physical Unit Block Extension (PUBX)

The PUBX table is a logical extension of the PUB table. There is one PUBX entry for each device added at IPL. A PUBX entry is addressed via address table APBXAREA at offset  $4 * \text{PUB index}$  (see Figure below). The PUBX entries have variable length and contain device related information. The PUBX area is preceded by the eye catcher PUBXTAB.

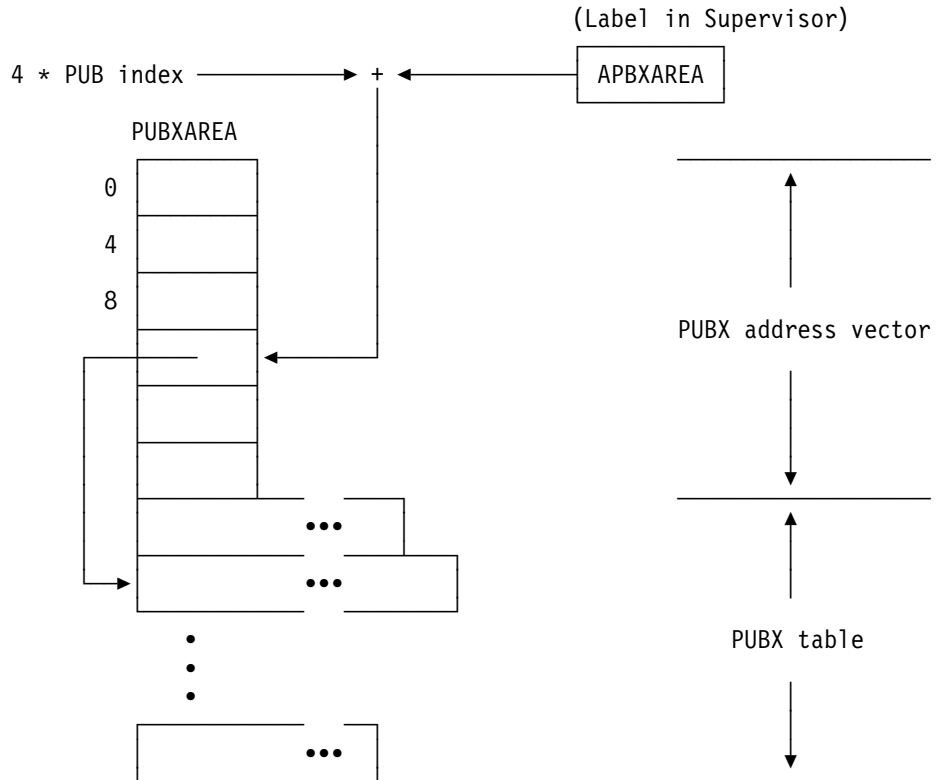


Figure 213. PUBX Table Interrelationship

Bytes		Label	Description
Dec	Hex		
0	0	PBXFLAG	Flag byte
		PBXDASD	X'80' DASD device
		PBXTAPE	40 Tape device
		PBXUR	20 Unit record device
			10 Reserved
			08 Reserved
		PBXMSSDV	04 Mass storage device
		PBXFBAVD	02 Virtual disk device (FBA)
		PBXSLOG	01 SYSLOG device
1	1	PBXFLAG1	Flag byte
		PBXSHR	X'80' Partition sharable device DASD, SYSLOG and POWER spool UR
			40 Reserved
		PBXBUFFD	20 Buffered device
			DASD: Cached controller
		PBXFEAT1	10 TAPE: Data compaction
			DASD: Dasd fast write
		PBXFEAT2	08 TAPE: AUTO BLOCKING
			DASD: Cache Fast Write
		PBXFEAT3	04 TAPE: Library online
			DASD: Dual copy capable
			UR: Reserved
		PBXNORDB	02 TAPE: NO read backward
			DASD/UR: Reserved
		PBXDBCCW	01 Set Mode command with data
2	2	PBXFLAG2	Flag byte 2 Device status
		PBXNOSIO	X'80' SIO needs interception
		PBXMTFLG	X'40' Mount request pending
		PBXNOSVT	X'20' Bypass AVR processing
		PBXNOASS	X'10' Bypass (UN)ASSIGN-CCW
		PBXNOVOL	X'08' VOL1 label not yet read
			X'04' Reserved
			X'02' Reserved
		PBXREADY	X'01' Device became ready
3	3	PBXFLAG3	Flag byte 3 : Processing Flag
4 - 5	4 - 5	PBXCUU	CUU address
6 - 7	6 - 7	PBXSCH	Subchannel number
8 - 9	8 - 9	PBXLEN	Length of PUBX entry
10 - 11	A - B	PBXVTMCQ	Index of dechained CHANQ entry
12 - 19	C - 13	PBXCHPID	Channel path IDs
20 - 36	14 - 24	PBXDVNED	Initial device NED
37	25	PBXPIM	Path Installed Mask
38	26	PBXPAM	Path Available Mask
39	27	PBXLPM	Logical Path Mask
40	28	PBXNOOPR	Not oper. path mask (due to OFFLINE)
41	29	PBXNOQSD	Not oper. path mask (due to Quiesce)
42	2A	PBXNOPVF	Not oper. path mask (due to path verification)
43	2B	PBXNOERP	Not oper. path mask (due to perm err.)
44	2C	PBXVPVEN	Used internally
45 - 51	2D - 33		Reserved
52	34	PBXRCCA	Channel Connection Address
53	35	PBXRDCDC	Director Device Connection
54 - 55	36 - 37		Reserved
56 - 63	38 - 3F	PBXSIMAD	Used internally
64	40	PBXPUBCD	VSE device type code
65 - 71	41 - 47	PBXSNSID	Sense device type information
65	41		X'FF' Entry is valid
			X'FE' Entry is generated

Figure 214 (Part 1 of 2). Physical Unit Block Extension (PUBX)

Bytes		Label	Description
Dec	Hex		
66 - 67	42 - 43	PBXCUTYP	Control unit type number
68	44	PBXCUMOD	Control unit model number
69 - 70	45 - 46	PBXDV TYP	Device type number
71	47	PBXDV MOD	Device type model number
72 - 73	48 - 49		Reserved
74 - 75	4A - 4B	PBXOBR	OBR record id
76	4C	PBXMDR	MDR record id
77	4D	PBXCUID	CU model identifier
78 - 79	4E - 4F	PBXOWNER	PIK of partition owning the device, in case of non-sharable device
.....			
80 - 83	50 - 53	PBXUSCNT	(if PBXSHR OFF) Device usage counter
84 - 87	54 - 57	PBXJACNT	Job Accounting SIO counter
.....			
80 - 83	50 - 53	PBXUSOFF	(if PBXSHR ON) Offset of usage counters within partition SIO counter table
84 - 87	54 - 57	PBXJAOFF	Offset of SIO counters within partition SIO accounting table
.....			
88 - 91	58 - 5B	PBXERBLK	Address of error block
92 - 95	5C - 5F	PBXDOMID	Operator Message ID
96 - 111	60 - 6F	PBXIRB	IRB information from last I/O
(112)	(70)	PBXCLNG	Length of common section
.....End of section common to all devices.....			
112- 115	70 - 73	PBXCCW	Address of prefix CCW's
116- 119	74 - 77		Reserved
120- 123	78 - 7B	PBXVCTE	Address of VCT entry
(124)	(7C)	PBXDLNG	Length of DASD section
.....End of section for DASD devices .....			
124- 127	7C - 7F	PBXMODE	Mode information for tapes
124	7C	PBXACMD	Currently valid mode set command
125	7D	PBXABYT	Currently valid mode set data byte
126	7E	PBXSCMD	Saved permanent mode set command
127	7F	PBXSBYT	Saved permanent mode set data byte
128	80	PBXTREC	Special recording information
		PBXT7TRK	X'80' Media is a 7-track tape
		PBXTCTGK	40 Media is a tape cartridge
			20 - 04: Reserved
		PBXT2XF	02 3490 XF-2 Format
			01 Reserved
129	81	PBXTDEN	Recording densities capability
			80 Reserved
			40 Reserved
			20 Reserved
			10 Reserved
		PBXT6250	08 6250 BPI
		PBXT3200	04 3200 BPI
		PBXT1600	02 1600 BPI
		PBXT0800	01 0800 BPI
130- 131	82 - 83	PBXTSPEC	Tape Features
132- 135	84 - 87	PBXLBINF	Tape Library information or zero
(136)	(88)	PBXTLNG	Length of TAPE section
.....End of section for TAPE devices .....			

Figure 214 (Part 2 of 2). Physical Unit Block Extension (PUBX)

## Physical Unit Block Table 2 (PUB2)

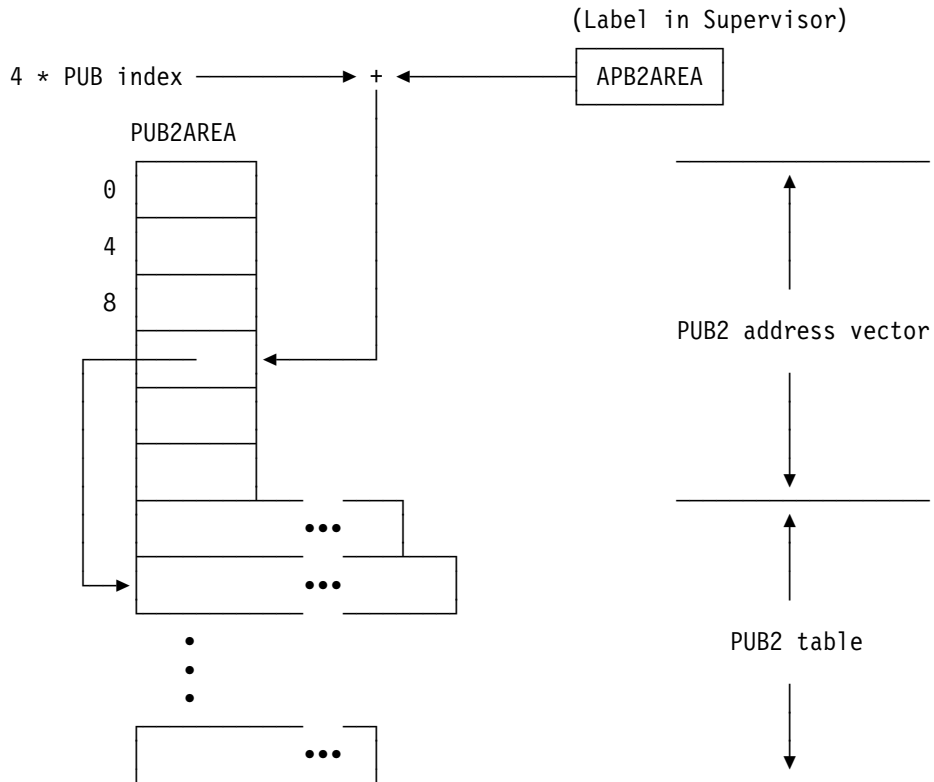


Figure 215. PUB2 Relationship

Bytes		Label	Description
Dec	Hex		
0 - 2 3	0 - 2 3	P2USAGE P2FLAGS P2INTSM P2DIAGM P2NORCM P2STAT2 P2NAMEF P2OPEN P2NOSIO P2DOMID	Usage count (number of non-ERP SIO) Flag byte common to all PUB2 entries X'80' Device is in intensive mode 40 Device is in diagnostic mode 20 No recording mode 10 Call statistics transient 2 08 Use PUB2 name completion field 04 Volume opened on this device 02 Intercept next SIO 01 Delete operator message
4	4	P2LIMIT	CE mode limit byte
5	5	P2BBMASK	CE mode byte/bit mask
(6)	(6)	PUB2EXT	End of fixed part of PUB2 Start of PUB2 extension

Figure 216. Physical Unit Block Table 2 (PUB2)

## PUB Ownership Table (PUBOWNER)

Bytes 120 - 123 (X'78'-X'7B') of the System Communication Region (SYSCOM) contain the address of the PUB Ownership Table. Label PUBOWNER identifies the first byte of this table. Each entry consists of a string of 16 bits, with each bit representing an owner due to the following layout.

**B'1000000000000000'** device is owned by VTAM

**B'010xxxxxxxxxxxxx'** device is owned by at least one dynamic partition

**B'0x01xxxxxxxxxxxx'** device is owned by System.

The other bits correspond to the static partitions in the sequence of:

**B'0x0xxxxxxxxxxxxx1'** BG

**B'0x0xxxxxxxxxxxxx1x'** FB

**B'0x0xxxxxxxxxxxxx1xx'** FA

**B'0x0xxxxxxxxxxxxx1xxx'** F9

**B'0x0xxxxxxxxxxxxx1xxxx'** F8

**B'0x0xxxxxxxxxxxxx1xxxxx'** F7

**B'0x0xxxxxxxxxxxxx1xxxxxx'** F6

**B'0x0xxxxxxxxxxxxx1xxxxxxx'** F5

**B'0x0xxxxx1xxxxxxx'** F4

**B'0x0xxx1xxxxxxx'** F3

**B'0x0xx1xxxxxxx'** F2

**B'0x0x1xxxxxxx'** F1

**Note:** In case a dynamic partition is owning a device (bit 1 is set), an extension to the appropriate device owner entry does exist. This extension is allocated separately and consists of a bitstring with as many bits as dynamic partitions have been specified at IPL time. Each bit in this extension represents a single dynamic partition with the lowest bit (bit 0) being used for that dynamic partition with the lowest PIK value.

## Device Usage Counters (DVCUSCNT)

For devices, which are not partition sharable (PBXSHR=0), the usage and SIO counters are included in the PUBX, see Figure 214 on page 480. For partition sharable devices (PBXSHR=1), one set of usage and SIO counters is needed for every partition. All usage counters belonging to one partition are allocated as a string. The address of the string can be found in PCB.PCBCNT. The offset of the usage counters of a given device within the partition string can be found in fields PUBX.PBXUSOFF and PUBX.PBXJAOFF.

The SIO counter for Job Accounting is a single 4-byte field. For partition sharable devices, SIO counters are included in the partition string only if SYS JA=YES was specified at IPL time.

Device usage counters are always allocated. Their structure and meaning are described below.

Bytes		Label	Description
Dec	Hex		
0-1	0-1	DVCPUCNT	(if DVCPWRSP OFF) Physical usage counter Gives the number of times a device is physically accessed in a partition, either via a Logical Unit assignment or via physical addressing.
0-1	0-1	DVCPWRD	(if DVCPWRSP ON) This field contains the TID of the task, which has a spooling request pending. If no request is pending, it contains X'0000'.
2	2	DVCLUFLG	Flag byte
		DVCPWRSP	X'80' Reserved 40 Used as a dummy device for POWER 20 Reserved 10 Reserved 08 Reserved 04 Reserved 02 Reserved 01 Reserved
2-3	2-3	DVCLUCNT	Logical usage counter Gives the total number of Logical Unit assignments to this device within a partition.

Figure 217. Device Usage Counters (DVCUSCNT)



## Channel Control Table (CHNTAB)

Label CHNTAB identifies the first byte of the Channel Control Table. The Channel Control Table is just being left for compatibility reasons.

Bytes		Label	Description
Dec	Hex		
0	0	CHNTYPE	Unused
1	1	CHNTERR	Number of unit checks pending on this channel
2	2	CHNTFLG1	Processing Flag Byte
		CHNRSTRT	X'80' Channel must be restarted
		CHNRSDEV	40 At least one device not started
			20 Reserved
			10 Reserved
			08 Reserved
			04 Reserved
			02 Reserved
			01 Reserved
3	3	CHNTFLG2	Channel ID
4-7	4-7	CHNTPUBF	Address of first PUB on channel
8-11	8-B	CHNTPUBN	Reserved
12-15	C-F		Reserved

Figure 218. Channel Control Table (CHNTAB)

## Channel Queue Table (CHANQ)

Bytes 37-39 (X'25'-X'27') of the System Communication Region (SYSCOM) contain the address of the Channel Queue Table. Label CHANQ identifies the first byte of the Table. Each entry is fixed length and its layout is as follows:

Bytes		Label	Description
Dec	Hex		
0	0	CHQCHAIN	Index of next entry in free list or device queue.
0-3	0-3	CHQCCBAD	Address of CCB/IORB associated with I/O request
4	4		Reserved
5	5	CHQPROC	Logical processing flag required
		CHQDOINT	X'80' Interrupt not yet processed
		CHQDQUNC	40 Dequeue unconditional
		CHQNODEQ	20 Do not dequeue entry
			10 Reserved
			08 Reserved
		CHQDASFP	04 DASD file protect needed
		CHQFILE	02 SYSFIL on CKD device
		CHQSFFBA	01 SYSFIL on FBA device
6	6	LUBDSP	System logical unit number
			X'FF' for Programmer logical units
7	7	TKREQID	Task ID (TID) of request owner
8	8	CHQCCSIO	SIO flag byte
		CHQCCACT	X'80' Device is active
			40 Reserved
		CHQCCPRI	20 Primary channel I/O
		CHQCCLTE	10 Long time entry (Missing Interrupt Handler)
		CHQCCRUN	08 Condition Code 0
		CHQCCSW	04 Condition Code 1
		CHQCCBSY	02 Condition Code 2
		CHQCCNOP	01 Condition Code 3
9	9	CHQCCBB1	Copied from byte 2 of CCB/IORB
10	A	CHQCCBB2	Copied from byte 3 of CCB/IORB
11	B	CHQCCBB3	Copied from byte 12 of CCB/IORB
12	C	CHQPFIX	Reserved for page fixing routine.
13-15	D-F	CHQPFIXL	Address of user specified or internal fixlist

Figure 219 (Part 1 of 2). Channel Queue Table (CHANQ)

Dec	Bytes		Label	Description
	Hex			
16	10		CHQERRCT	Error retry count
17	11		CHQCHNHI	High order byte of next entry in chain
18	12		CHQPUBHI	High order byte of PUB index
19	13		CHQPUBNO	PUB entry number
20	14		CHQFLG1	Flag byte
				X'80' Reserved
			CHQHQA	40 Head queue request
			CHQCSBSY	20 Device busy saved from PUB
			CHQCSQED	10 Device queued-in-error from PUB
			CHQDIDJA	08 Request was already accounted
				04 Reserved
				02 Reserved
			CHQFSI01	01 Start on primary channel only
21	15		CHQGRP	Requestor flag
			CHQGROLT	X'80' OLTEP request
			CHQGRBTM	40 BTAM request
			CHQGRVTM	20 VTAM request
				10 Reserved
			CHQRRAS	08 RAS request
			CHQRRROK	04 Successful retry
				02 Reserved
			CHQPVTOK	01 PVT has been initiated
22	16		CHQDEV	Device group indicator
			CHQDASD	X'80' CKD device or diskette
			CHQFBA	40 FBA device
			CHQTAPE	20 TAPE device
			CHQTP	10 TP (teleprocessing) device
			CHQCRT	08 2260 or 3277 device
			CHQURC	04 Unit record device
				02 Reserved
				01 Reserved
23	17		CHQIOINF	Delayed interrupt exit indicator
				X'00' Dispatcher
				04 I/O initiator (INITRG)
				08 I/O interrupt handler (INTRTRN)
				0C Error ignore routine (IGNORE)
				10 Cancel with code X'1A' (ERR1A)
				14 Reserved
				18 Dequeue routine (DEQUNCON)
				1C Post routine (PSTRESET)
				20 Emergency MSG writer
24-31	18-1F		CHQCSW	Accumulated interrupt status

Figure 219 (Part 2 of 2). Channel Queue Table (CHANQ)



Count	Trans- mission Informa- tion	CSW Status Bits	Type Code and Logical Unit	Used by LIOCS or 3895 PIOCS	CCW Addr.	Used by Physical IOCS	CCW Address in CSW	Optional Sense CCW							
0	1	2	3	4	5	6	7	8	9	11	12	13	15	16	23
Byte(s)				Description											
2-3 (cont.)				Byte 3											set on by
				Bit 0: DASD Data Check in Count Area, 3350 permanent error, 3203, PRT1 or 5203 Print Check, Equipment Check 3540 Special Record transferred.											PIOCS
				Bit 1: DASD Track Overrun, PRT1 Print Quality/Equipment check											PIOCS
				Bit 2: DASD End-of-Cylinder, PRT1/2245 Line position error.(Note 7)											PIOCS
				Bit 3: 2520, 2540 or 3881 Equipment Check, 3203, 5203 Data check, Equipment Check 3505 or 3525 Permanent Error, (Note 8) TAPE Read Data check, DASD Data Check, PRT1 Print Check/Data Check, Diskette Data Check.											PIOCS
				Bit 4: CARD Unusual command sequence, DASD No Record Found, PRT1 UCSB, PRT1 UCSB Parity Check (Command retry),											PIOCS
				Bit 5: User does not expect NO RECORD FOUND condition.											Pr.Pr.
Note:				Pr.Pr. stands for Problem Program											

Figure 220 (Part 2 of 4). Command Control Block (CCB)

Count	Trans- mission Informa- tion	CSW Status Bits	Type Code and Logical Unit	Used by LIOCS or 3895 PIOCS	CCW Addr.	Used by Physical IOCS	CCW Address in CSW	Optional Sense CCW							
0	1	2	3	4	5	6	7	8	9	11	12	13	15	16	23
Byte(s)		Description													
2-3 (cont.)		Byte 3												set on by	
		Bit 6: PRINTER Carriage Channel 9 Overflow, DASD Verify error												PIOCS	
		Bit 7: Channel Program is not retryable Retry will be started from failing CCW if possible.												Pr.Pr.	
4-5 CSW STATUS BYTES		Byte 4						Byte 5 (Note 1)							
		Bits:						Bits:							
		0 (32): Attention						0 (40): Program Controlled Interruption							
		1 (33): Status Modifier						1 (41): Incorrect Length							
		2 (34): Control Unit End						2 (42): Program Check							
		3 (35): Busy						3 (43): Protection Check							
		4 (36): Channel End						4 (44): Channel Data Check							
		5 (37): Device End						5 (45): Channel Control Check							
		6 (38): Unit Check						6 (46): Interface Control Check							
		7 (39): Unit Exception						7 (47): Chaining Check							
6-7 TYPE code and UNIT ID		Byte 6													
		B'1x0xx0xx' = User-translated CCB													
		B'01x0x00x' = BTAM CCB													
		B'0x1xx0xx' = System-translated CCB													
		B'x0xx10yy' = CCB for physical unit (yy is high order bits of 10 bit PUB index with byte 7 containing low order bits)													
		B'xxxx0001' = CCB for program logical unit													
		B'xxxx0000' = CCB for system logical unit													
Note:		Pr.Pr. stands for Problem Program													

Figure 220 (Part 3 of 4). Command Control Block (CCB)



2. Indicates /\* or /& statement read on SYSRDR or SYSIPT. byte 4, bit 7 (Unit Exception) is also on.
3. DASD data checks on count not returned.
4. The traffic bit (Byte 2, bit 0) is normally set on at channel end to signify that the I/O was completed. If byte 2, bit 5 has been set on, the traffic bit and bits 2 and 6 in byte 3 will be set on at device end. See also Note 1.
5. This error occurs as an equipment check, data check or FCB parity check. For 2245, this error occurs as a data check or FCB parity check.
6. Byte 2, bit 6 must be set on to allow you to accept 3505, 3525 permanent errors. This bit is forced on by LIOCS if the user specified ERROPT for his input or output files. Byte 3, bit 3 is set on if a permanent error was encountered.
7. If User Error Routine is specified and the user needs the sense information to further process the error, byte 12, bit 2 must also be set. Otherwise, the supervisor error routine will clear off the status on return and the sense information is not available.
8. 3895 error codes are returned in CCB byte 8. Refer to *3895 Document Reader/Inscriber Machine and Programming Description* for information on these error codes.



## Input/Output Request Block (IORB)

The IORB establishes communication between the problem program and physical IOCS. The IORB consists of a fixed length part (24-bytes) and some optional extension fields each of it fixed length (4-bytes), which are all appended to each other.

Bytes		Description
Dec	Hex	
0- 1	1- 1	Residual count, Number of bytes which were not transferred by the channel
2	2	Communication Byte 1 Set by Physical IOCS: X'80' WAIT Bit, Traffic Bit (Note 1) X'40' End-of-File on SYSRDR or SYSIPT, /* or /& (Note 2)  X'20' Permanent I/O error encountered The following bits can be set by problem program X'10' Prevent Cancellation in case of permanent I/O Error X'08' Reserved X'04' User wants to be posted at device end time (Note 1)  X'02' Reserved X'01' Skip system error Recovery (no Recovery Action)
3	3	Communication Byte 2 Reserved for ERP return information.
4	4	Device Status Information (Note 3) X'80' Attention X'40' Status modifier X'20' Control unit end X'10' Busy X'08' Channel end X'04' Device end X'02' Unit check X'01' Unit exception
5	5	Channel Status Information (Note 3) X'80' Program controlled Interrupt X'40' Incorrect length X'20' Program check X'10' Protection check X'08' Channel data check X'08' Channel control check X'02' Interface control check X'01' Channel Chaining check

Figure 221 (Part 1 of 3). Input/Output Request Block (IORB)

Bytes Dec      Hex		Description																											
6-7 TYPE code and UNIT ID		Byte 6 B'1xxxx1xx' = Reserved B'01xxx1xx' = Reserved B'001xx1xx' = System-translated IORB B'00xx11yy' = IORB for physical unit (yy is high order bits of 10 bit PUB index with byte 7 containing low order bits) B'00xx0101' = IORB for program logical unit B'00xx0100' = IORB for system logical unit																											
6-7 TYPE code and UNIT ID		Byte 7 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 33%;">Byte 6 bit 4 off + 7 off</th> <th style="width: 33%;">Byte 6 Bit 4 off 7 on</th> <th style="width: 33%;">Byte 6 Bit 4 on</th> </tr> </thead> <tbody> <tr> <td>SYSRDR=00    SYSRLB=08</td> <td>SYS000=00</td> <td>PUB entry No.</td> </tr> <tr> <td>SYSIPT=01    SYSUSE=09</td> <td>SYS001=01</td> <td>000</td> </tr> <tr> <td>SYSPCH=02    SYSREC=0A</td> <td>SYS002=02</td> <td>.</td> </tr> <tr> <td>SYSLST=03    SYSCLB=0B</td> <td>.</td> <td>.</td> </tr> <tr> <td>SYSLOG=04    SYSDMP=0C</td> <td>.</td> <td>.</td> </tr> <tr> <td>SYSLNK=05    SYSCAT=0D</td> <td>.</td> <td>.</td> </tr> <tr> <td>SYSRES=06    SYSLUB=0E-FF</td> <td>.</td> <td>.</td> </tr> <tr> <td>SYSSLB=07</td> <td>SYS255=FF</td> <td>3FF (bits 6-7 of byte 6 used)</td> </tr> </tbody> </table>	Byte 6 bit 4 off + 7 off	Byte 6 Bit 4 off 7 on	Byte 6 Bit 4 on	SYSRDR=00    SYSRLB=08	SYS000=00	PUB entry No.	SYSIPT=01    SYSUSE=09	SYS001=01	000	SYSPCH=02    SYSREC=0A	SYS002=02	.	SYSLST=03    SYSCLB=0B	.	.	SYSLOG=04    SYSDMP=0C	.	.	SYSLNK=05    SYSCAT=0D	.	.	SYSRES=06    SYSLUB=0E-FF	.	.	SYSSLB=07	SYS255=FF	3FF (bits 6-7 of byte 6 used)
Byte 6 bit 4 off + 7 off	Byte 6 Bit 4 off 7 on	Byte 6 Bit 4 on																											
SYSRDR=00    SYSRLB=08	SYS000=00	PUB entry No.																											
SYSIPT=01    SYSUSE=09	SYS001=01	000																											
SYSPCH=02    SYSREC=0A	SYS002=02	.																											
SYSLST=03    SYSCLB=0B	.	.																											
SYSLOG=04    SYSDMP=0C	.	.																											
SYSLNK=05    SYSCAT=0D	.	.																											
SYSRES=06    SYSLUB=0E-FF	.	.																											
SYSSLB=07	SYS255=FF	3FF (bits 6-7 of byte 6 used)																											
8	8	Reserved																											
9-11	9- B	Virtual address of the CCW associated with this IORB																											
12	C	Reserved for physical Input Output Control System (PIOCS) X'80' IORB is used by Error Recovery Procedure X'40' Reserved X'20' This IORB has an extension X'10' Reserved X'08' Reserved X'04' Reserved X'02' Reserved X'01' Reserved																											
13-15	D- F	Address+8 of last CCW that was executed																											

Figure 221 (Part 2 of 3). Input/Output Request Block (IORB)

Bytes		Description
Dec	Hex	
16	10	Fix Flag X'80' Fix List is already in compressed format (Each page to be fixed for Channel Program execution is covered only once within the FIXLIST) X'40' All pages are FIXED (The user has already fixed all the pages need for channel program execution) X'20' Reserved X'10' Reserved X'08' Reserved X'04' Reserved X'02' Reserved X'01' Reserved
17-19	11-13	Address of FIXLIST
20-21	14-15	IORB Version identification code
22-23	16-17	Special processing flags set by LIOCS Bit 0 SYSFIL request for FBA Device Bits 1-15 Reserved
.....		
OPTIONAL		
24	18	Parameter ID: Bit 0 IDentifies the last optional Parameter Bits 1-7 Parameter ID  B'0000000' ECB ID B'xxxxxxX' Reserved
25-27	19-1B	Address portion of optional Parameter
.....		
		•
		•
		•
		•
		Parameter ID:
.....		
		Parameter ID:

**Notes:**

1. The WAIT Bit (byte 2, bit 0) is normally set on at Channel End to signify that at least the data transfer is completed.  
If byte 2, bit 5, has been set on, the WAIT Bit is set at Device End.
2. Unit Exception (Byte 4, bit 7) is also turned on.
3. Bytes 4 and 5 contain the status bytes of CSW (Bits 32-47) which is always the accumulated status information received so far.

Figure 221 (Part 3 of 3). Input/Output Request Block (IORB)

## Disk Information Block (DIB) Tables

DIB Table for CKD and DISKETTE  
 DIB for FBA Device  
 DIB Extension (DIBX) Table (required by FBA)

There is one DIB table per partition. Its address can be found at label PCEADIB in the partition's PCE.

The DIB tables for the static partitions are allocated during supervisor generation. Each DIB table comprises a number of single entries. There is one entry for each, SYSLNK (open information), SYSIN, SYSPCH and SYSLST.

The DIB entries for dynamic partitions are allocated in the system getvis area during dynamic partition allocation and consist of a single for SYSLNK only.

### Disk Information Block (DIB) Table for CKD and DISKETTE

There are different formats of the DIB entries:

Current Record Address	Length of Key	Data	End of Extent Address	Head High	No. Low	Max Rec.	Notify Rec.No.	Flag	
0	6	7 8 9	10	16	17	18	19	20 21	22 23
Byte(s)		Description							
0	CURRENT RECORD ADDRESS	Specifies the disk address of the next sequential record. The format differs slightly depending on the file and the device. For CKD-devices:                      For DISKETTE-devices: SYSIN : BBCCHHR                      SYSIN : 0000CHR SYSPCH : BBCCHHR                      SYSPCH : 0000CHR SYSLST : BBCCHHR                      SYSLST : 0000CHR							
7	KEY LENGTH	Always zero							
8	DATA LENGTH	Data length of record to be processed For CKD-devices:                      For DISKETTE-devices: SYSIN : X'0050' or X'0051'                      SYSIN : X'0000' SYSPCH : X'0051'                      SYSPCH : X'0000' SYSLST : X'0078'                      SYSLST : X'0000'							
10	END of EXTENT ADDRESS	Specifies the disk address of the last record within the given Extent. For CKD-devices:                      For DISKETTE-devices: SYSIN : BBCCHHR                      SYSIN : 0000CHR SYSPCH : BBCCHHR                      SYSPCH : 0000CHR SYSLST : BBCCHHR                      SYSLST : 0000CHR							
Note: The DIB is initialized by Job Control with Extent Info. and updated by PIOCS on every I/O oper. to the appropriate device.									

Figure 222 (Part 1 of 2). Disk Information Block Table (DIB) for CKD Devices and Diskette

Current Record Address	Length of Key	Data	End of Extent Address	Head No. High	Low	Max Rec.	Notify Rec.No.	Flag					
0	6	7	8	9	10	16	17	18	19	20	21	22	23
Byte(s)		Description											
17	HIGHEST HEAD NO.	Highest head number accessible on this device											
18	LOWEST HEAD NO.	Lowest head number accessible on this device											
19	MAXIMUM NO. of RECORDS	Maximum number of records that fit on one track											
20	NOTIFY RECORD NUMBER	This field specifies the number of records that the user wants to be checked at EOJ time of whether they still fit into the specified Extent (applicable for output only). This field is set by the JCL SET statement (RCLST or PCPCH). A warning message will be issued when this minimum number has been reached or exceeded during the previous JOB.											
22	FLAG BYTE	Flag byte: X'40' Device with RPS feature											
23	RESERVED	Not used											
Note: The DIB is initialized by Job Control with Extent Info. and updated by PIOCS on every I/O oper. to the appropriate device.													

Figure 222 (Part 2 of 2). Disk Information Block Table (DIB) for CKD Devices and Diskette

## Disk Information Block Table (DIB) for FBA Device

Bytes		Label	Description
Dec	Hex		
0-3	0-3	ULPBN	End address of extent. Upper limit of physical block number
4-7	4-7	CRPBN	Current address. Current physical block number
8-9	8-9	CIOFF	Offset of current record within control interval
10-11	A-B	LNGCI	Length of control intervals in bytes
12	C	PBPERCI	Number of physical blocks per control interval
13-15	D-F	PBUFFER	Pointer to data buffer
16	10	DIBFLAGS	X'80' DIB gate flag X'40' Task waiting for DIB X'20' Reserved X'10' Source begin readjustment required X'08' Reserved X'04' Force write out X'02' End of extent reached X'01' Buffer-in-use flag
17-19	11-13	PDIBX	Pointer to DIB extension (DIBX)
20-21	14-15	DIBRSCNT	Residual count for JCL message
22-23	16-17		Reserved

Figure 223. Disk Information Block Table (DIB) for FBA Devices

The FBA device also requires a DIB Extension (DIBX) Table.

Bytes		Description
Dec	Hex	
0-23	0-17	Input Output Request Block (IORB)
24-31	18-1F	Fixlist first area
32-39	20-27	Fixlist second area
40-47	28-2F	DEFINE EXTENT CCW
48-55	30-37	LOCATE CCW
56-63	38-3F	READ/WRITE CCW
64-79	40-4F	DEFINE EXTENT Parameter list
80-87	50-57	LOCATE Parameter list

Figure 224. DIB Extension Table (DIBX) for FBA Devices

## ERBLOC Area

The ERBLOC is an area in the SGERP part of the supervisor used as an interface between system components involved in I/O Error- or Recording Request processing. The AERBLOC field (Bytes 0-3) of the System Communication Region (SYSCOM) contains a pointer to the ERBLOC area. Internal macro ERBLOC maps the ERBLOC.

Bytes		Label	Description
Dec	Hex		
0- 7	0- 7	SVC5NM	Name of first/next ERP Transient to be fetched or used last
8-11	8- B	YRETRY	Continuation address for I/O RETRY
12-15	C- F	YIGNORE	Continuation address for error IGNORE
16-19	10-13	ACANCEL	Continuation address for I/O CANCEL
20-23	13-17	YERPEXIT	Common DSK/ERP return address
..... Begin of Unit Check entry .....			
24-31	18-1F	ERQCSW1	Channel Status Word at time of error
32-33	20-21	ERRPUB1	PUB pointer of affected device
34	22	ERQFLG1	Flag Byte
		TRUNRF	X'80' No record found on DASD 40 Set device interv. required 20 Pass back error information
		IGNERR	10 Force error IGNORE exit
		SUCCESS	08 Error successfully recovered
		RTYERR	04 Force SIO RETRY exit 02 Force Recording only
		OCCUP	01 Error block is in use
35	23	ERQMSG1	X'E2' Soft error X'AE' Reserved for recording X'xx' Message code
36-39	24-27	ERQSEK1	Used for disk devices only CKD: Failing Seek address FBA: OS device type codes
40	28	ERQCCB1	Index of channel queue entry X'FF' for unsolicited error CCB pointer (0 if no CCB available)
41-43	29-2B		
44-75	2C-4B	ERQSNS1	Sense data
..... End of UNIT CHECK entry .....			
..... Layout of RECORDING entry .....			
24-27	18-1B	ERQAEADR	SD record address
28	1C	ERQAELEN	Length of SD record
29	1D	ERQAETYP	Type of SD record X'FF' = Record builder request
30	1E	ERQAESW1	Record dependent switch 1
31	1F	ERQAESW2	Record dependent switch 2
32-33	20-21	ERQAEPUB	PUB pointer of affected device
34	22	ERQAEFLG	Flag Byte X'80' SD record is TFIX-ed 02 Must be 0 for recording info.
		OCCUP	01 Error entry is in use
35	23	ERQAEMSG	Contains X'AE' for Alternate Entry
36-39	24-27	ERQAETIB	TIB of requesting task
40	28	ERQAEIND	Set to FF
41-43	29-2B		Reserved
44-75	2C-4B	ERQAECOM	Communication information
..... End of RECORDING entry.....			

.....ERBLOC continued .....			
76-79	4C-4F	ERQPUB21	PUB2 pointer
80-83	50-53	ERQCAW1	CAW for RETRY as prepared by ERP
84-91	54-5B	ERQPRIN1	ERP processing information
84	54	ERQLSNS1	Number of sense bytes
85	55	ERQPRFL1	Processing flags
		ERQPSOV	X'80' Multiple unit checks
		ERQPVT	X'40' Path verification required
			X'20' Reserved
		ERQV510	X'10' Extended ERBLOC version
		ERQPMSG	X'08' Message processing only
			X'04' Reserved
			X'02' Reserved
			X'01' Reserved
86	56	ERQERPF1	ERP communication flags
			X'80' Reserved
		ERQPOBR	X'40' OBR processing
		ERQPMDR	X'20' MDR processing
		ERQPSIR	X'10' ALERT processing
		ERQPPER	X'08' Permanent error
		ERQPTM	X'04' Temporary error
		ERQPSOF	X'02' Soft error
			X'01' Reserved
87	57	ERQPATH1	Path failure information
88-91	58-5B	ERQCOMM1	ERP communication area
92-95	5C-5F	ERQPUBX1	PUBX pointer
96	60	ERQACPU1	Processor ID information
97	61		Reserved
98	62	ERQERR1	Error count
99	63	ERQCCOD1	Command Code set by ERP
100-103	64-67	ERQCHQA1	Reserved
104-111	68-6F	ERQFJOB1	Failing job name
112-115	70-73	ERQSCSW1	Subchannel status word 0
116-119	74-77	ERQESW1	Extended status word 0
120-123	78-7B	ERQPUBN1	Pub index
124-125	7C-7D	ERQTID1	Task ID of failing task
126-127	7E-7F	ERQRFRL1	Length of recording area
128-129	80-81	ERQRFRA1	Address of recording area
130-305	82-131	ERQRFRC1	RF record area
306-321	132-141	ERCHNOFT	Chain header offset table, used to address the following error chains
322-325	142-145	RASERCHN	Address of first RAS error entry
326-329	146-149		Pointer to RAS TIB
330-333	14A-14D	ERPERCHN	Address of first ERP error entry
334-337	14E-151		Pointer to ERP TIB
338-341	152-155	DSKERCHN	Address of first DSK error entry
342-345	156-159		Pointer to DSK TIB
346-349	15A-15D	SNSERCHN	Address of first SNS error entry
350-353	15E-161		Pointer to SNS TIB
354-389	162-185	SNSSDAID	Sense data saved by SDAID

**Note:** See Figure 226 on page 501.

Figure 225 (Part 2 of 2). ERBLOC Area



## I/O Error Entry

There is one I/O error entry for each device. Field PBXERBLK in the PUBX contains a pointer to this entry. An additional error entry exists for some system tasks. The address of this entry is contained in field TCBERBLK of the system task TCB.

Bytes		Label	Description
Dec	Hex		
0-3	0-3	ERBLKPTR	Pointer to next error entry in a chain or 0
4	4	ERBLKFLG	Flag byte
		HQERBLK	X'80' System task error entry
		ALTCHANN	40 Error on alternate channel
		ERSNSDAV	20 Sense data available
		ERACTIVE	10 Error entry active
		ERQUEUED	08 Error entry is enqueued in some error chain
			04 Reserved
			02 Reserved
			01 Reserved
5	5	ERBLKFLG1	Flag byte
			X'80' Reserved
			40 Reserved
			20 Reserved
			10 Reserved
		NEEDSNS	08 Must be processed by SNS task
		NEEDDSK	04 Must be processed by DSK task
		NEEDERP	02 Must be processed by ERP task
		NEEDRAS	01 Must be processed by RAS task
6	6		Reserved
7	7	ERBLKSNL	Number of sense bytes
8 - 11	8 - B	ERRQCHQA	Address of channel queue entry
12 - 43	C - 2B	ERRQE4ID	Extended Sense ID information
..... End of error entry header .....			

Figure 226 (Part 1 of 3). I/O Error Recovery/Recording Entry

Bytes		Label	Description
Dec	Hex		
.....		Unit Check	error entry .....
44	2C	ERRQLSNS	Number of sense bytes
45	2D	ERRQPRFL	Processing flag
		ERRQPSOV	X'80' Multiple unit checks
		ERRQPVT	X'40' Path verification required
			X'20' Reserved
			X'10' Reserved
		ERRQMSG	X'08' Message processing only
		ERRQBDPM	X'04' Duplex pair suspension message
			X'02' Reserved
			X'01' Reserved
46	2E	ERRQERPF	ERP communication flag
			X'80' Reserved
		ERRQPOBR	X'40' OBR processing
		ERRQPMDR	X'20' MDR processing
		ERRQPSIR	X'10' ALERT processing
		ERRQPPER	X'08' Permanent error
		ERRQPTM	X'04' Temporary error
		ERRQPSOF	X'02' Soft error
		ERRQPT91	X'01' Type 91 MDR
47	2F	ERRQPATH	Path failure information
48 - 51	30 - 33	ERRQCOMM	ERP communication area
52 - 59	34 - 3B	ERRQCSW	CSW indicating error condition
60 - 61	3C - 3D	ERRQPUB	PUB pointer of affected device
62	3E	ERRQFLG	Flag Byte
		TRUNRF	X'80' No record found on DASD
			40 Reserved
		PASSBK	20 Pass back error
		IGNERR	10 IGNORE is possible
		SUCCESS	08 Error successfully recovered
		RTYERR	04 Channel command RETRY possible
		RECONLY	02 Force recording only
		OCCUP	01 Error entry is in use
63	3F	ERRQMSG	Message Code
			X'E2' Soft error
		RCVMSG	X'20' Initial value in case
			Sense command failed
64 - 67	40 - 43	ERRQSEK	Used for disk devices only
			CKD: Failing Seek address
			FBA: OS device type codes
68	44	ERRQCQPT	Index of channel queue entry
			X'FF' for unsolicited error
68 - 71	44 - 47	ERRQCCB	CCB pointer (0 if no CCB available)
72 - ..	48 - ..	ERRQSNS	Sense data
.....		End of UNIT CHECK entry .....	

Figure 226 (Part 2 of 3). I/O Error Recovery/Recording Entry

..... Channel Check entry .....			
44 - 45	2C - 2D	CCEQNSNS	Reserved
46	2E	CCEQFLG	channel check handler Flag byte
		CCSIO	X'80' Channel Check on SIO
		CCDAM	X'40' Channel damage
			X'20' Reserved
			X'10' Reserved
		CCREC	X'08' Record built or written
			X'04' Reserved
		CCDSK	X'02' Channel check on DASD device
			X'01' Skip message writer
47	2F	CCEQDMC	Channel failure information
48	30		Reserved
49	31	CCEQRTC	Retry counter
50	32	CCEQTIDN	Failing task id
52 - 59	34 - 3B	CCEQCSW	CSW at time of error
60 - 61	3C - 3D	CCEQPUB	PUB pointer of affected device
62	3E	CCEQSFLG	Flag Byte
			X'80' Reserved
			40 Reserved
			20 Reserved
			10 Reserved
			08 Reserved
			04 Reserved
			02 Reserved
		OCCUP	01 Error entry is in use
63	3F	CCEQMSG	Message Code
64 - 67	40 - 43	CCEQIOEL	I/O extended logout area address
68	44	CCEQCQP	Chanq Index of failing request
			X'FF' for unsolicited error
69 - 71	45 - 47	CCEQECSW	Extended Channel Status Word (ECSW)
.....End of CHANNEL CHECK entry .....			

Figure 226 (Part 3 of 3). I/O Error Recovery/Recording Entry

## Recorder File Table (RFTABLE)

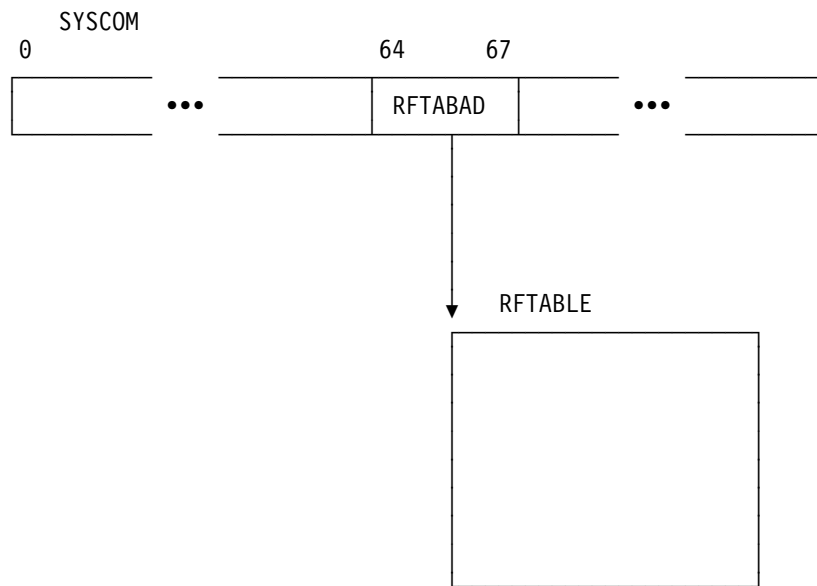


Figure 227. Recorder File Table Relationship

Bytes		Label	Description
Dec	Hex		
0	0	RFTABLE	Label of Starting Address
		RFFLAGS1	Flag byte 1
		RFFULL	X'80' File full
		RFRDE	40 RDE option included
		RFIPL	20 Initial IPL
		RFNO	10 RF=No option
		RFCREATE	08 File is to be created
		RFBUILT	04 File has been created
		RFONFBA	02 File on FBA device
		RFREADY	01 File ready
1	1	RFFLAGS2	Flag byte 2
		FFMSG	X'80' File full message request
		LTMSG	40 Last track message request
		IEMSG	20 I/O error message request
		DLMSG	10 Data lost message request
		RFEVA	08 EVA message request
		RFRTAOWN	04 File owned by RTA recorder
		RFPTAOWN	02 File owned by PTA recorder
		RFEREP	01 File being accessed by EREP
		2	2
LTMISUD	X'80' Last track msg issued once		
RECDERR	40 Error is to be recorded		
RECDSF	20 Short form record request		
RFIRULT	10 Individual records for unlabeled tapes		
	08 Reserved		
RFHIOERR	04 Error in writing RFHEADER		
RFBOMT05	02 Exit to \$\$BOMT05 indicator for \$\$BOPEN		
RFBOMT01	01 Exit to \$\$BOMT01 indicator for \$\$BOPEN		
3	3		
			X'80' - X'02' Reserved
4	4	RFRNW	01 No record written
		RFFLAGS5	Flag byte 5
5	5	RFFLG5BD	X'80' - X'02' Reserved
		RFNOFN	01 BOPEND called by OPEN
6	6	RFRECTYP	N of N for records (low order 4 bits contain the number of records to be recorded and high order 4 bits contain the number of the records being recorded)
		RFRELTYP	Record type code
7	7	RFREL	Release level code of VSE/Adv.Function
8	8	RFRDSW1	Record dependent byte 1
		RFTEMP	X'40' Temporary error

Figure 228 (Part 1 of 2). Recorder File Table (RFTABLE)

Bytes		Label	Description
Dec	Hex		
9	9	RFRDSW2	Record dependent byte 2
10 - 11	A - B	RFBUFLG	Length of data buffer (FBA)
..... CKD		Device Related	Information .....
18 - 19	12 - 13	RFRECLEN	Length of record
20	14	RFRDSW3	Record dependent switch 3
21	15		Reserved
22 - 23	16 - 17	RFRCLCKD	CKD Block length
24 - 27	18 - 1B	RFRECADR	Address of record
28 - 34	1C - 22	RFSEEK	Work area for seek addr.BBCCHHR
28 - 29	1C - 1D	RFSEEKBB	BB portion of seek
30 - 31	1E - 1F	RFSEEKCC	CC portion of seek
32 - 33	20 - 21	RFSEEKHH	HH portion of seek
34	22	RFSEEKR	R portion of seek
35	23		Reserved
36 - 39	24 - 27	RFHDRCH	SYSREC cylinder/head
36 - 37	24 - 25	RFHRCYL	Cyl. address of file start
38 - 39	26 - 27	RFHDRTRK	Head address of file start
..... End of CKD		Device Related	Information .....
..... FBA		Device Related	Information .....
12 - 15	C - F	RFBUFAD	Address of data buffer
16 - 17	10 - 11	RFNAVR	Displacement of next available RDF in buffer (FBA)
18 - 19	12 - 13	RFRECLEN	Length of record
20	14	RFRDSW3	Record dependent switch 3
21 - 23	15 - 17		Reserved
24 - 27	18 - 1B	RFRECADR	Address of record
28 - 31	1C - 1F	RFCUBL	Work area for block number
32 - 35	20 - 23		Reserved
36 - 39	24 - 27	RFHDRBL	SYSREC block number of file start
..... End of FBA		Device Related	Information .....
40 - 41	28 - 29	RFCHMAP	Map of supported channels
42 - 49	2A - 31	RFCHIDC	Channel ID codes
50 - 51	32 - 33	RFEREPID	EREP Tash ID for EOTSK
52 - 55	34 - 37	RFEXIT	Exit phase name or exit address
56	38	RFEVARTH	EVA read threshold
57	39	RFEVAWTH	EVA write threshold
58 - 59	3A - 3B	RFP2ENTL	Length of PUB2 table
60 - 63	3C - 3F	RFP2ENT	Address of PUB2 table
64 - ...	40 - ...	RFP2ITAB	PUB2 index table (see Note)

**Note:** Two bytes are generated for each PUB2 index entry.  
See also Figure 215 on page 482.

Figure 228 (Part 2 of 2). Recorder File Table (RFTABLE)

## Machine and Channel Check Control Blocks

RAS Linkage Area (RASLINK)  
RAS Monitor Table (RASTAB)  
Error Recovery Procedure Information Block (ERPIB)

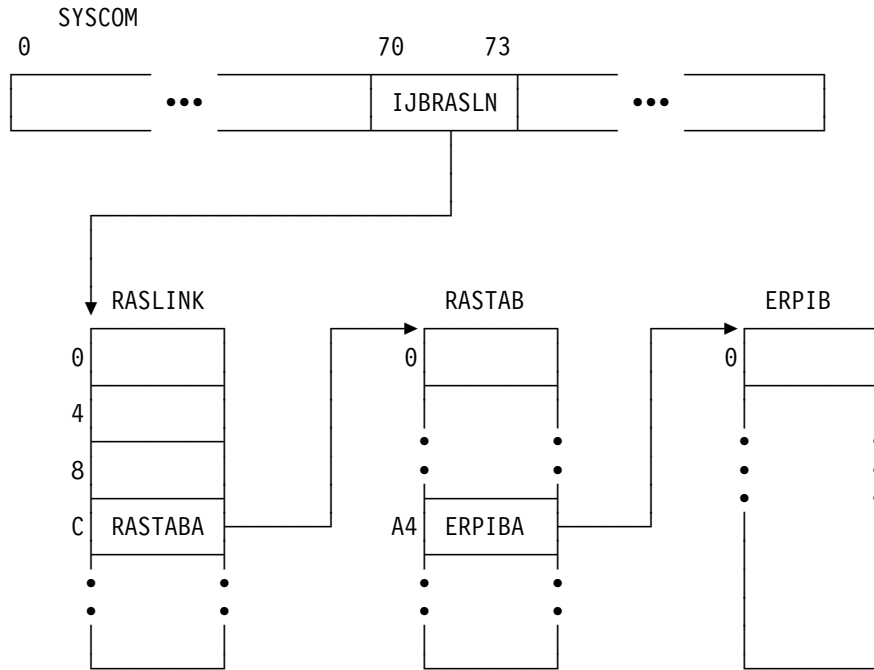


Figure 229. Machine/Channel Check Control Block Relationship

## RAS Linkage Area (RASLINK)

Bytes		Label	Description
Dec	Hex		
0-3	0-3	CPUIDW1	First part of CPUID field
4-7	4-7	CPUIDW2	Second part of CPUID field
5	5	CPUID	Model number in CPUID field
6	6	RASMCELL	Length of machine check extended logout area
8	8	RASDMC	Damaged channel ID
9	9	RASFLAGS	RAS flag byte
		RASACT	X'80' RAS task activated
		RASMCCT	40 Machine check handling
		RASCCACT	20 Channel check handling
		RASCRWAC	10 CRW Handling in progress
		RASEMGEX	08 Emergency handling
		RASSTERM	04 System termination
			02 Reserved
		RTAIOA	01 RAS task I/O active
10	A	MCFLAGS	Machine check flags
		MCHARD	X'80' Hard machine check
		MCEXTD	X'08' External damage to be processed
		MCCRWP	X'04' CRW pending request to process
		MCEVIP	X'02' Event information pending
		MCDLYP	X'01' Delayed MC processing pending
11	B	RASRSFLG	RAS recording status flag
		RASDEBUG	X'80' RAS Debug mode
			40 Reserved
		RASNOMSG	20 Unrecoverable channel check on SYSLOG
			10 Reserved
			08 Reserved
		RASBTDEQ	04 BTAM dequeue request
		RASMSGRT	02 Return from RAS message writer
		RASMSGIO	01 RAS message I/O
12-15	C-F	RASTABA	Address of RAS monitor table (RASTAB)
16-19	10-13	RASBASE	RAS base address
20-21	14-15	RASIMOD	Internal model number
22-23	16-17	RASIOELL	Length of I/O extended logout area
24-27	18-1B	RASMCELA	Address of machine check extended logout area
			X'80' Indicates field contents not valid
28-31	1C-1F	RASTAREA	Address of RAS transient Area
32-35	20-23	RAS\$PUBX	Address of PUBX area
36-47	24-2F	RASPGRID	Path group ID (IPL)
48	30	RASCPVER	Real CPU version code (SPDT)
49-51	31-33		reserved
52-55	34-37	RASCHSCB	Channel Subsystem contl.block address
56-59	38-3B	RASMSTK	Machine Check stack area address

Figure 230. RAS Linkage Area (RASLINK)



## RAS Monitor Table (RASTAB)

Bytes		Label	Description
Dec	Hex		
0-3	0-3	RASMCERQ	Machine Check Error Queue
4-7	4-7	RASCRWNX	Addr of next CRW to process
8	8	TPAFLAG	TPA CC Recovery Flag
9-103	9-67		reserved
104-115	68-77	RASCCB	RAS CCB
116-147	78-97	RASCCWS	RAS CCW chain
148-154	98-9E	RASEEK	Seek address of RAS seek
155	9F	RTAOWN	R-transient identifier
156-159	A0-A3		reserved
160-163	A4-A7	ERPIBA	Address of work ERPIB
164-167	A8-AB		reserved
168	AC	RTAID	Requestor ID for RTA I/O
		RASRECID	X'08' RAS recording request
		RASRTYID	X'04' Channel retry request
		RASRTYXR	X'02' Retry for EXCP REAL (set together with RASRTYID)
169	AD	ERPID	Return load index for WTOR
170-171	AE-AF	RASRES	Device address of SYSRES
172-173	B0-B1	RASREC	Device address of SYSREC
174-175	B2-B3	RASLOG	Device address of SYSLOG
176-243	B4-F3	TRANSAV	RTA register save area, Register 0 to Register 15
244-307	F4-133	SYSREGS	RAS monitor register save area, Register 0 to Register 15
308-311	134-137	SUPLINK	Service routine address for RTA in RAS monitor
308	134	LINKFLAG	Flag byte indicating requested service
		RASLIO	X'80' Perform normal I/O
		RASLEMIO	40 Perform emergency I/O
		RASLFTCH	20 Fetch another transient
		RASLWAIT	10 Perform wait
		RASLPDEQ	08 Dequeue page frame
		RASLDEQ	04 Dequeue CCB/IORB
		RASLFREE	02 Free I/O extended logout area
			01 reserved
		RASLEXIT	00 Exit from RAS transient
312-323	138-143	HIR	Hardware instr. retry accumulator
312-313	138-139	HIRACNT	Accumulated HIR count
314-315	13A-13B	HIRLCNT	Threshold value for count
316-319	13C-13F	HIR1TME	Time of day for first error of group
320-323	140-143	HIRLTME	Time threshold value in timer units
324-335	144-14F	ECCMAIN	Main storage error accumulators
324-325	144-145	ECCACNT	Accumulated ECC count for main stor.
326-327	146-147	ECCLCNT	Threshold value for count
328-331	148-14B	ECC1TME	Time of day for first error of group
332-335	14C-14F	ECCLTME	Time threshold value in timer units

Figure 231 (Part 1 of 2). RAS Monitor Table (RASTAB)

Bytes		Label	Description
Dec	Hex		
336-347	150-15B	MACHERR	Machine Check accumulators
336-337	150-151	MCERRCNT	Accumulated machine check count
338-339	152-153	MCLCNT	Threshold for count
340-343	154-157	MC1TIME	Time of day for first error
344-347	158-15B	MCLTIME	Time threshold in clock units
348	15C	MCMODE	Hardware operation mode
349	15D	BUFDEL	Count of buffers deleted
350	15E	RASMSG1	RAS Message byte 1
		MCPR	X'80' Power restored - running normal
		MCPL	40 Power lost - running UPS
		*	20 reserved
		MTICLDMG	10 Clock and or timer damage
		MTIMDMG	08 Timer damage
		*	04 reserved
		MPPERFDEG	02 System performance degradation
		MEFLOVFL	01 EFL overflow
351	15F	RASMSG2	RAS Message byte 2
		MCLOKDMG	X'80' Clock damage, all modes quiet
		MLASTTR	40 Threshold on recorder file reached
		MPAGEDEL	20 Buffer pages deleted
			10 reserved
			08 reserved
		MFILEFL	04 Recorder file full
		MUNRCIO	02 Error on recorder file
		MCRECOV	01 Successful recovery from machine check
352	160	RASMSG3	RAS Message byte 3
			all bits reserved
353-355	161-163		reserved
356-357	164-165	RASIND	RAS indicators
		RASNODEQ	X'80' Page frame not dequeued
358-363	166-16B		Reserved
364-383	16C-17F	INTERSEG	Interface segment build area
364-367	16C-16F	FXDLGADD	Address of fixed logout area
368-371	170-173	FXDLGADD	Address of extended logout area
372	174	INOFN	Sequence number: record one of n
373	175	ILOGL	Logout length in record one
374	176	IRECL	Total length of record one
375	177	NNOFN	Sequence number record n of n
376	178	NLOGL	Logout length in record n
377	179	NRECL	Total length of record n
378-379	17A-17B		reserved
380-384	17C-180	MACHIND	Indicate machine check

Figure 231 (Part 2 of 2). RAS Monitor Table (RASTAB)

## Error Recovery Procedure Information Block (ERPIB)

Bytes		Label	Description
Dec	Hex		
0-7	0-7	ERPIBCSW	Saved channel status word
0	0	ERPIBSTC	ERPIB status codes
		ERPIBEND	X'FF' Indicate end of ERPIB
		ERPIBFRE	X'FE' Indicate free ERPIB
		ERPIBCNC	X'FD' Indicate task is to be canceled
		ERPIBCCR	X'FC' Indicate retry unsuccessful
		ERPIBCCS	X'FB' Indicate retry successful
0-3	0-3	ERPIBCCW	Address of failing CCW + 8
4	4	ERPIBST1	First status byte
5	5	ERPIBST2	Second status byte
6-7	6-7	ERPIBCNT	Residual count in CSW
8-11	8-B	ERPIBIOE	Ptr to corresp. I/O ext. logout area
12	C		Reserved
13	D	ERPIBDMC	Damaged channel ID
14-15	E-F	ERPIBPUB	PUB address of failing device
16	10	ERPIBCQP	Channel queue present indication
17	11	ERPIBRTC	RAS retry counter
18	12	ERPIBMSG	Message indicator
		ACTMSG	X'80' Wait for operator response
		CCDONE	40 Channel check handling complete
		CCNODEQ	20 PUB not queued in error
			10 Reserved
			08 Reserved
			04 Reserved
		RECC	03 Recovered channel check
		ERRCC	02 Channel check
		HRDCC	01 Unrecoverable channel check
19	13	ERPIBREQ	Requestor ID
20	14	ERPIBFLG	Flag byte
		CCSIO	X'80' Channel check on SIO
		CCDAM	40 Channel damage
			20 Reserved
			10 Reserved
		CCREC	08 Record build or written
			04 Reserved
		CCDSK	02 Channel check on disk device
		CCSKM	01 Skip message writer
21-23	15-17	ERPIBESW	Extended CSW
21-22	15-16		
23	17	ERPIBSC	ERPIB sequence code
23	17	ERPIBTC	ERPIB termination code
			X'80' Reserved
			40 Reserved
		CCAREP	20 Channel check ancillary report
24-27	18-1B		reserved
28-31	1C-1F	ERPIBCQA	Channel queue address
32-39	20-27	ERPIBTOD	TOD clock
40-43	28-2B	ERPIBQA	Addr of CC Stack entry (3590 TPA)
44-47	2C-2F		reserved
		ERPIBWIT	Length of ERPIB

Figure 232. Error Recovery Procedure Information Block (ERPIB)

## Track Hold Table (THTAB)

Bytes 76-79 (X'4C' - X'4F') of the System Communication Region (SYSCOM) contain both, the free list pointer and the address of the Track Hold Table. Label THFLPTR identifies the free list pointer and label THTAB identifies the first byte of the table.

The halfword at THTAB-2 contains the total number of 16-byte entries comprising the track hold table.

Bytes		Label	Description
Dec	Hex		
0	0	THPTR	Index of next entry in the chain (forward pointer) X'FF' indicates last entry
1-3	1-3	THCCB	Address of CCB/IORB
4-11	4-B	THTRK	CKD devices: Address of track in BBCCHH00 format FBA devices: Physical block numbers of first and
12	C	THBWPTR	Index of previous entry in the chain (backward pointer)
13	D	THFLG	Flag and count byte X'80' Another task is waiting for this track/block 40 First entry within a PUB chain 20 Reserved 10 Reserved
14-15	E-F	THCTR THTID	Bits 4-7: NO. of concurrent holds - 1 Task ID of track/block owner

Figure 233. Track Hold Table (THTAB)

---

## Appendix C. Samples

### *Track Hold Processing*

Figure 234 on page 514 shows the initialized significant bytes of the track hold mechanism.

Figure 235 on page 515 illustrates the pointers and table entries after several track hold requests have been issued.

Figure 236 on page 516 summarizes the sequence of events leading to the situations shown in Figure 235 on page 515, Figure 237 on page 517, and Figure 238 on page 518.

When a task requests a hold on a track/block that is already held by another task, the high-order bit of the flag-and-counter byte is turned on (for example, entry No. 1 in Figure 235 on page 515). When a task requests a hold on a track/block it holds itself, the flag-and-counter byte is incremented by one (for example, entry No. 4 in Figure 235 on page 515).

On release of a track/block by the holding task, and provided the counter is zero before the release, any task or tasks that are waiting for that track/block are brought out of the wait state. The supervisor then returns to task selection and, if the next selected task was waiting for this track/block, its hold request is honored. Any other task or tasks that were waiting for the track/block now remain ready-to-run, but if such a task gains control before the track/block has again been released, that task returns to the wait state.

If the counter is not zero before the release, then only the counter is decremented by one so that the track/block remains held by the same task. For illustrations of these operations, compare Figure 235 on page 515 and Figure 237 on page 517, entries number one and four.

### Track Hold Table Entries

Free List Pointer THFLPTR	Entry No.	Chain Byte	CCB/IORB Addr.	BBCCHH00 or LLPBN+ULPBN	Backward Pointer	Flag and Counter	Task ID
00 →	0	01	zeros	zeros	zeros	zeros	zeros
	1 ←	02	zeros	zeros	zeros	zeros	zeros
	2 ←	03	zeros	zeros	zeros	zeros	zeros
PUBS Track Hold Pointers (PUBOPTN)	3 ←	04	zeros	zeros	zeros	zeros	zeros
	4 ←	05	zeros	zeros	zeros	zeros	zeros
	5 ←	06	zeros	zeros	zeros	zeros	zeros
FF	6 ←	07	zeros	zeros	zeros	zeros	zeros
FF	7 ←	08	zeros	zeros	zeros	zeros	zeros
FF	8 ←	09	zeros	zeros	zeros	zeros	zeros
FF	9 ←	FF	zeros	zeros	zeros	zeros	zeros
FF							

**Note:**

**THFLPTR:** The track hold free list pointer (1 byte) contains a pointer to the first entry in the free list or X'FF' when the track hold table is full.

**BBCCHH00:**

- 00,
- cylinder cylinder,
- head head,
- 00

**LLPBN:** Low limit physical block number

**ULPBN:** Upper limit physical block number

*Figure 234. Track Hold Table Example. Initial contents of significant bytes used by track hold requests.*

Track Hold Table Entries

Free List Pointer THFLPTR	Entry No.	Chain Byte	CCB/IORB Addr.	BBCCHH00 or LLPBN+ULPBN	Backward Pointer	Flag and Counter	Task ID
05	0	01	xxx	Track 1A	PUB ptr.	40	aa
	1	02	xxx	Track 1B	00	80	aa
	2	04	xxx	Track 1C	01	00	bb
PUBS Track HoldA	3	FF	xxx	Track 2A	PUB ptr.	40	aa
Pointers A (PUBOPTN) A	4	FF	xxx	Track 1D	02	40	aa
A 2nd Dev.▶▶	5	06	zeros	zeros	zeros	zeros	zeros
03	6	07	zeros	zeros	zeros	zeros	zeros
FF	7	08	zeros	zeros	zeros	zeros	zeros
FF	8	09	zeros	zeros	zeros	zeros	zeros
1st Device ◀◀00	9	FF	zeros	zeros	zeros	zeros	zeros
FF							

Figure 235. Track Hold Table Example. Task aa holding tracks 1A, 1B, and 1D (2 holds) on 1st device, and track 2A on 2nd device; task bb holding track 1C on first device; a task is waiting to hold track 1B on first device.

Sequence of Requests	Tasks			Remarks
	aa	bb	cc	
↓	Hold 1A	Hold 1C		Entry queued
	Hold 1B			Entry queued
	Hold 2A			Entry queued
	Hold 1D			Entry queued
	Hold 1D			Counter incremented
			Hold 1B	Entry flagged and requester put into wait state.

The table entries and pointers at this stage are illustrated by Figure 235.

↓	Free 1B			Flag turned off, waiting task (cc) made ready-to-run, and task selection entered; if task cc is selected, its request for track 1B is honored
	Free 1A			Entry dequeued.
	Free 1D			Counter decremented.

The table entries and pointers at this stage are illustrated by Figure 237 on page 517.

↓	Free 2A			Entry dequeued.
	Free 1D			Entry dequeued.
		Free 1C		Entry dequeued.
			Free 1B	Entry dequeued.

All tracks have now been freed as shown in Figure 238 on page 518.

Figure 236. Example of Tracks Held and Freed by Three Tasks



Track Hold Table Entries

Free List Pointer THFLPTR	Entry No.	Chain Byte	CCB/IORB Addr.	BBCCHH00 or LLPBN+ULPBN	Backward Pointer	Flag and Counter	Task ID
00	0	05	xxx	Track 1A	zero	00	00
	1	FF	xxx	Track 1B	04	00	cc
PUBS Track HoldA	2	04	xxx	Track 1C	PUB ptr.	40	bb
Pointers A	3	FF	xxx	Track 2A	PUB ptr.	40	aa
(PUBOPTN) A	4	01	xxx	Track 1D	02	00	aa
2nd device A	5	06	zeros	zeros	zeros	00	00
03	6	07	zeros	zeros	zeros	00	00
FF	7	08	zeros	zeros	zeros	00	00
FF	8	09	zeros	zeros	zeros	00	00
1st Device	9	FF	zeros	zeros	zeros	00	00
02							
FF							

Figure 237. Track Hold Table Example. Task aa has released holds on tracks 1B and 1A, and one of the holds on track 1D; task cc has been taken out of the wait state and has been selected to run, so it now holds track 1B.

### Track Hold Table Entries

Free List Pointer THFLPTR	Entry No.	Chain Byte	CCB/IORB Addr.	BBCCHH00 or LLPBN+ULPBN	Backward Pointer	Flag and Counter	Task ID
01	0	05	xxx	Track 1A	00	00	00
	1	02	xxx	Track 1B	00	00	00
	2	04	xxx	Track 1C	00	00	00
PUBS Track Hold Pointers A (PUBOPTN) A	3	00	xxx	Track 2C	00	00	00
	4	03	xxx	Track 1D	00	00	00
	5	06	zeros	zeros	00	00	00
2nd Device FF	6	07	zeros	zeros	00	00	00
FF	7	08	zeros	zeros	00	00	00
FF	8	09	zeros	zeros	00	00	00
1st Device FF	9	FF	zeros	zeros	00	00	00
FF							

Figure 238. Track Hold Table Example. Situation after total release.

---

## Appendix D. XPCC/APPCVM Protocol

### Introduction

XPCC/APPCVM protocol may be used in a VM environment to communicate between a requestor-application, operating in a VSE virtual machine and a server application, operating in a CMS virtual machine.

The guest machines may be on different CPUs within one TSAF collection, or within different TSAF collections being part of an SNA network (see *Virtual Machine/System Product Transparent Services Access Facility Reference*, SC24-5287).

Requestor applications in VSE exchange data with server applications in CMS via VSE-XPCC. The VSE-XPCC services are extended to XPCC/APPCVM to support communication from a VSE guest machine to a non-VSE guest machine in a VM environment.

VSE XPCC/APPCVM is a subset of VSE XPCC. The VSE application is the communication requestor - the User Program in TSAF terms -, and the other VM machine is the server - the Resource Manager in TSAF terms.

The VSE application that wants to communicate with an application in another VM machine has to use the limited set of VSE XPCC services listed below.

- IDENTify
- CONNECT
- SENDR with/without data to be sent
- CLEAR
- DISCONNect
- DISConnect/PuRGe
- TERMINate

Internally the VSE-XPCC requests are transformed to APPCVM requests. The data to be sent have to be placed into the data buffer in APPC logical record format, and the data received are in APPC logical record format.

The VSE XPCC/APPCVM support is activated at IPL if VSE is initialized as a VM guest.

## XPCC/APPCVM Activation

The VSE XPCC/APPCVM support is activated when VSE IPL is performed on a VM virtual machine. Activation means, that the VSE guest machine is enabled for APPCVM external interrupts. APPCVM connections are not established, before a VSE application requests the begin of APPCVM communication.

APPCVM communication can be established in two ways:

- At VSE IPL connection routing information is supplied. The IPL SET command (described in *z/VSE System Control Statements*, SC33-8305) maps a VSE-application name to the APPCVM target (VM resource). The XPCC CONNECT service uses the VSE to application name as an alias, searches the mapping information supplied by the IPL SET command, and - if found - it builds an APPCVM connection using the APPCVM information.
- The XPCCB TOAPPL operand names the APPCVM target (VM resource) directly. This is indicated by the new parameter FDSCR=APPCVM in the XPCC CONNECT request.

No VSE IPL connection routing information is required within a TSAF collection. If the VM resource is linked by a VM gateway, then the SNA network routing information has to be supplied by an IPL SET command. The XPCC CONNECT service searches the information supplied by the IPL SET command, and - if found - it builds an APPCVM connection using the network routing information.

## Overview on XPCC Functions Supported as XPCC/APPCVM

Input change to XPCC commands:

1. Limited protocol.
2. No user data supported.
3. Data in APPC logical record format.
4. Minimum reply buffer length 2 bytes.

Additional output from XPCC commands:

1. Service state indicator.
2. New return information.

### Function Restriction

Only a subset of XPCC functions is supported. The application may use the following XPCC functions:

- IDENTify
- CONNECT
- SENDR with/without data to be sent
- CLEAR
- DISCONNect
- DISConnect/PuRGe
- TERMINate

The supported XPCC commands do not have any changed operands. However, there is a restriction the application has to be aware of. User data is not transferred to the communication partner. It is ignored.

User data is the data that may be placed by the application (requesting a XPCC function) into the XPCCB field IJBXSUSR. If the target is within the same VSE system, the user data is transferred immediately to the partner-application's XPCCB field IJBXRUSR. The XPCC/APPCVM restriction is introduced, because APPCVM does not support the transfer of user data.

### Data Format

The data placed into the XPCC send buffer has to be in APPC logical record format, and the data received in the XPCC reply buffer is in APPC logical record format.

Each APPC logical record has a 2-byte length field followed by a data field. The application has to insert the 2-byte length field in front of the data field of a record to be sent, and it has to remove it from data received.

The data field can range from 0 to 32,765 bytes long. The data in an XPCC send or reply buffer may consist of one or more complete APPC logical records, the beginning of a record, the middle of a record, the end of a record, or by complete records preceded by the end of a record and/or followed by the beginning of a record.

### Buffer Length

The length of a reply buffer may not be zero. It has to be at least 2 bytes long to accommodate the APPC logical record length which also is present for a data field of length zero.

## State Indicator

The APPCVM half duplex protocol forces each communication partner to assume a unique state. The connection requestor is initially in SEND state 'S' at connection completion. The connection partner (acceptor) is initially in RECEIVE state 'R'.

A SENDR request sending and receiving data may be issued in SEND state only. The application which is in SEND state loses its state with the SENDR request, and assumes the RECEIVE 'R' state. Now the communication partner is in SEND state. The application in RECEIVE state cannot obtain back the SEND state explicitly. It stays in RECEIVE state, until the communication partner issues a request to receive new data.

To indicate this internal protocol state to the XPCC application (which may not try to send data when in RECEIVE state), the internal APPCVM state is passed in XPCCB field IJBXSTAT.

As long as the link is pending but not yet active, or the link is severed (internally by APPCVM SEVER), the XPCCB field IJBXSTAT contains a state indicator 'N'(not active). The only XPCC command that may be issued in the 'N' state is XPCC DISCONN.

The values may be:

State	When state occurs	XPCC function that may be issued
SEND 'S'	After CONNECT completes - IJBXCECB posted. After SENDR completes - IJBXSECB posted. Condition: the partner replies, and wants to receive new data. After CLEAR completes - IJBXSECB posted.	SENDR with data CLEAR DISCONN DISCPRG
RECEIVE 'R'	After SENDR completes - IJBXSECB posted. Condition: the partner only replies.	SENDR without data CLEAR DISCONN DISCPRG
Link inactive 'N'	From CONNECT to CONNECT completion. After a function completes with SEVER, until DISCONN request.	DISCONN

## Error Information

Return codes, error codes and reason codes are given back as they are in the current XPCC services.

- R15 indicates successful or unsuccessful completion.
- XPCCB field IJBXRETC contains the information or error code describing the specific situation detected, if R15 is not equal zero.
- XPCCB field IJBXREAS contains the reason code describing the specific situation existing at asynchronous completion.

Additional error codes and additional reason codes are defined because of the APPCVM support. If errors are raised by the internally used APPCVM protocol, then the VSE error code or reason code indicates this. For debugging purposes, - or for applications with sophisticated error recovery routines - , the VM error information is also passed to the VSE application. It is copied by the XPCC/APPCVM service to the XPCCB user area field IJBXRUSR. Detailed explanations about the error situation can be found in the appropriate VM documentation (e.g. *Virtual Machine/System Product Transparent Services Access Facility Reference*, SC24-5287).

The format of the error information depends on the time when the error is detected. If it occurs immediately, then R15 and IJBXRETC contain XPCC error codes, and the immediate APPCVM code is returned in format A.

In case it occurs asynchronously, then IJBXREAS contains the XPCC reason code, and the APPCVM error information is returned in format B. Examples for asynchronous errors are:

- an APPCVM function completes with an error,
- the partner SEVERs the connection before a APPCVM function is complete,
- the partner SEVERs the connection without an APPCVM function pending,
- the partner violates the protocol with an unsupported APPCVM function.

The general layout of IJBXRUSR is the following:

Format A:

	ORG	IJBXRUSR	
IJBXVMFC	DS	CL1	APPC/VM FUNCTION CODE
IJBXVMRC	DS	CL1	THE APPC/VM ERROR CODE

Format B:

	ORG	IJBXRUSR	
IJBXVMFC	DS	CL1	APPC/VM FUNCTION CODE THE EXTERNAL
*			INTERRUPT IS ASSOCIATED WITH
IJBXVMEI	DS	0CL7	THE ERROR DATA OF APPC/VM EXTERNAL
*			INTERRUPT INFORMATION IN THE EIB
IJBXVTYP	DS	CL1	VM INTERRUPT INFORMATION IN "IPTYPE"
IJBXVCOD	DS	CL2	VM INTERRUPT INFORMATION IN "IPCODE"
IJBXVWRC	DS	CL1	VM INTERRUPT INFORMATION IN "IPWHATRC"
IJBXVAUD	DS	CL3	VM INTERRUPT INFORMATION IN "IPAUDIT"

- Field IJBXVMFC may contain a X'00' value if the interrupt occurs when no function is pending.
- Field IJBXVAUD contains X'000000' if the function having failed does not involve data buffers (e.g. internal SEVER, CLEAR/SENDERR).
- Fields IJBXVCOD, IJBXVWRC, IJBXVAUD contain X'00's if the interrupt type is SRI (send request interrupt).
- Fields IJBXVCOD, IJBXVWRC, IJBXVAUD contain X'00's if the interrupt is of a type, which is not expected, and cannot be handled.

**Note:** Specific values for the error information are described below with the services which can cause the errors to be presented.



## XPCC Functions not Supported with XPCC/APPCVM

Any XPCC function not belonging to the supported subset is rejected. The application will receive the following return code:

Return code	Value	Meaning
R15	X'08'	Request rejected. Specific error information is provided in the XPCCB field IJBXRETC.

The XPCC error code that may be set in the XPCCB control block at the same time with the return code X'08':

Field Name	Value	Meaning
IJBXRETC	IJBXIVFU	Invalid function. The function requested is not supported by XPCC/APPCVM.

## XPCC IDENTify Function

The IDENTify function does not imply any APPCVM services. However, a new restriction concerning naming conventions has to be observed.

In case the VSE XPCC/APPCVM support is active, then the subsystem name specified on the IPL SET XPCC command may not be used as identification name by a VSE application. This restriction is required, because the XPCC service would not know which subsystem to connect to, if the same subsystem existed within VSE and outside VSE. Special handling is performed with names beginning with character SYSARI, they are restricted for usage by SQL.

If the restriction above is not observed, the XPCC IDENTify request is rejected, and an error code is returned.

The XPCC IDENTify request is also rejected, if it is for a unique application name, and another application has identified itself with the same name; or if an application name is specified, with which an application has identified itself as unique application.

The errors are indicated as follows:

Return Code	Value	Meaning
R15	X'08'	Request rejected. Specific error information is provided in the XPCCB field IJBXRETC.

The XPCC error code that may be set in the XPCCB control block at the same time with the return code above:

Field Name	Value	Meaning
IJBXRETC	IJBXVMNM	VM subsystem name. Application name is invalid, because it is reserved as a VM subsystem name.
IJBXRETC	IJBXDUP	Duplicate application name. A unique application name is requested, but another application has already identified itself with the same name. Or an application name is requested, which has already been identified as unique name.

## XPCC TERMINate Function

The TERMINate function does not imply any APPCVM services.

## XPCC CONNECT Function

The CONNECT function implicitly uses the APPCVM CONNECT service. Thus errors can arise that reflect APPCVM problems. The errors may be indirect APPCVM problems detected by VSE or APPCVM problems analyzed by the VM support.

New error codes are:

Return Code	Value	Meaning
R15	X'08'	Request rejected. Specific error information is provided in the XPCCB field IJBXRETC.

The XPCC error codes that may be set in the XPCCB control block at the same time with the return code above:

Field Name	Value	Meaning
IJBXRETC	IJBXVMNA	APPCVM communication not activated. Communication to VM has not been initialized by an IUCV DCLBFR. (As noted by IPL message 0I81I.)
IJBXRETC	IJBXAPPC	APPCVM error. The internal APPCVM CONNECT request failed. The error information provided by APPCVM is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the error code IJBAPPC - has format A:

Field Name	Value	Meaning
IJBXVMFC	APPCVM CONNECT function code	The APPCVM function which is rejected.
IJBXVMRC	APPCVM IPRCODE value	An error occurred before the APPCVM CONNECT was initiated. The VM reason code is stored in IJBXVMRC. See APPCVM documentation.

New error codes given back when the CONNECT request was accepted and initiated, but its completion failed:

The XPCC reason code in the XPCCB control block has the following format:

Field Name	Value	Meaning
IJBXREAS	IJBXAPRS	APPCVM reason code. The internal APPCVM CONNECT request completed unsuccessfully. The APPC path is internally severed. The interrupt information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXUEXI	APPCVM reason code. The internal APPCVM CONNECT request terminated with an unexpected interrupt on this path. The APPC path is internally severed. The interrupt code presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXAPRS - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM CONNECT function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP, IJBXVCOD, IJBXVWRC	APPCVM EIB fields: IPTYPE, IPCODE, IPWHATRC	The APPCVM CONNECT request is rejected by the target partner (response was IUCV SEVER). The VM SEVER information contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXUEXI - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM CONNECT function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP	APPCVM EIB fields: IPTYPE	The APPCVM CONNECT request terminated with an unexpected interrupt. The VM interrupt code contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The application is in SEND state 'S' when the connection is complete indicated by posting of IJBXCECB.

State	When state occurs	XPCC function that may be issued
SEND 'S'	After CONNECT completes - IJBXCECB posted.	SENDR with data. DISCONN
Link inactive 'N'	After CONNECT is rejected - IJBXSECB posted. Condition: the partner responded with IUCV SEVER.	DISCONN

## XPCC DISCONNECT/DISCPRG Function

The DISCONNECT function implicitly uses the APPCVM SEVER service. Thus errors can arise that reflect APPCVM problems analyzed by the VM support.

New error codes are:

Return Code	Value	Meaning
R15	X'08'	Request rejected. Specific error information is provided in the XPCCB field IJBXRETC.

The XPCC error code that may be set in the XPCCB control block at the same time with the return code above:

Field Name	Value	Meaning
IJBXRETC	IJBXAPPC	APPCVM error. An internal APPCVM request failed. The error information provided by APPCVM is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the error code IJBXAPPC - has format A:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SEVER function code	The APPCVM function which is rejected.
IJBXVMRC	APPCVM IPRCODE value	An error occurred before the APPCVM SEVER was initiated. The VM reason code is stored in IJBXVMRC. See APPCVM documentation.

The application's state remains unchanged, but its value is irrelevant.

## XPCC SENDR Function

The SENDR function implicitly uses the APPCVM SENDDATA service, and the APPCVM RECEIVE service. Thus errors can arise that reflect APPCVM problems. The errors may be indirect APPCVM problems detected by VSE or APPCVM problems analyzed by the VM support.

New error codes given back when the SENDR request is rejected:

Return Code	Value	Meaning
R15	X'08'	Request rejected. Specific error information is provided in the XPCCB field IJBXRETC.

The XPCC error codes that may be set in the XPCCB control block at the same time with the return code above:

Field Name	Value	Meaning
IJBXRETC	IJBXPFIX	There is not sufficient real storage to fix the user's SEND or RECEIVE buffer.
IJBXRETC	IJBXINVS	Invalid state. A SENDR request was issued with IJBXBLN=0, but the application is in RECEIVE state 'R'. Or a SENDR request was issued with IJBXBLN=0, but the application is in SEND state 'S'.
IJBXRETC	IJBXAPPC	APPCVM error. The internal APPCVM SENDDATA or RECEIVE request failed. The error information provided by APPCVM is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the error code IJBXAPPC - has format A:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDDATA or APPCVM RECEIVE function code	The APPCVM function which was rejected.
IJBXVMRC	APPCVM IPRCODE value	An error occurred before the APPCVM SENDDATA or the APPCVM RECEIVE was initiated. The VM reason code is stored in IJBXVMRC. For explanation see APPCVM documentation.

New reason codes given back when the SENDR request was accepted and initiated, but its completion failed, or when the SENDR request is terminated because of path disconnection.

The XPCC reason code in the XPCCB control block has the following format:

Field Name	Value	Meaning
IJBXREAS	IJBXABCP	APPCVM reason code. The internal APPCVM SENDDATA or RECEIVE request completed unsuccessfully. The error information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXAPRS +IJBXABDC	APPCVM reason code. The internal APPCVM SENDDATA and RECEIVE function was terminated by a SEVER request from the communication partner. The APPC path is internally severed. The information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXUEXI	APPCVM reason code. The internal APPCVM SENDDATA and RECEIVE request was terminated by an unexpected interrupt on this path. The APPC path is internally severed. The information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXAPPV	APPCVM Protocol violation. The internal APPCVM SENDDATA or RECEIVE request was completed by an APPCVM request from the communication partner violating the supported protocol. The APPC path is internally severed. The information presented by APPCVM with the external interrupt - caused by the unsupported request - is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXABCP - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDDATA or APPCVM RECEIVE function code	The APPCVM function for which the interrupt occurs.



Field Name	Value	Meaning
IJBXVMEI Subfields: IJBXVTYP, IJBXVCOD, IJBXVWRC, IJBXVAUD	APPCVM EIB fields: IPTYPE, IPCODE, IPWHATRC, IPAUDIT	An error occurred when the APPCVM SENDDATA or APPCVM RECEIVE completed. The VM error information contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXAPRS+IJBXABDC - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDDATA or APPCVM RECEIVE function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP, IJBXVCOD, IJBXVWRC	APPCVM EIB fields: IPTYPE, IPCODE, IPWHATRC	A SEVER occurred when the APPCVM SENDDATA or APPCVM RECEIVE completed. The VM information contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXUEXI - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDDATA or APPCVM RECEIVE function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP	APPCVM EIB fields: IPTYPE	An unexpected interrupt occurred before the APPCVM SENDDATA or APPCVM RECEIVE completed. The VM interrupt code contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The APPCVM completion information in the XPCCB field IJBXRUSR - set with the reason code IJBXAPPV - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDDATA or APPCVM RECEIVE function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP, IJBXVCOD, IJBXVWRC	APPCVM EIB fields: IPTYPE, IPCODE, IPWHATRC	An APPCVM SENDDATA or APPCVM RECEIVE completed with a protocol violation from the other side. The VM information contained in the external interrupt data is stored in IJBXVMEI. If the external interrupt type is SRI, then IPTYPE is the only data presented. For explanation see APPCVM documentation.

The application stays in SEND state 'S' until the request is complete indicated by posting of IJBXSECB. After that the RECEIVE state 'R' is assumed, when the partner application only sends reply data. The SEND state 'S' is kept, when the partner application requested sending and receiving data.

State	When state occurs	XPCC function that may be issued
SEND 'S'	After SENDR completes - IJBXSECB posted. Condition: the partner replies, and wants to receive new data.	SENDR with data CLEAR DISCONN DISCPRG
RECEIVE 'R'	After SENDR completes - IJBXSECB posted. Condition: the partner only sends reply data.	SENDR without data CLEAR DISCONN DISCPRG
Link inactive 'N'	After SENDR completes unsuccessfully - IJBXSECB posted. Condition: the partner responded with SEVER.	DISCONN

## XPCC CLEAR Function

The CLEAR function implicitly uses the APPCVM SENDERR service. Thus errors can arise that reflect APPCVM problems. The errors may be indirect APPCVM problems detected by VSE or APPCVM problems analyzed by the VM support.

The function does not free buffers immediately as it does in a pure VSE environment. The application will receive the return code RC=4 indicating that the application may not yet reuse its data buffers. It has to wait until the XPCCB field IJBXSECB is posted; then the buffers are available again and the link is ready for the next XPCC SENDR request (application is in SEND state).

New return code given back when the CLEAR request is initiated:

Return Code	Value	Meaning
R15	X'04'	Request accepted. Specific information is provided in the XPCCB field IJBXRETC.

The XPCC error code that may be set in the XPCCB control block at the same time with the return code above:

Field Name	Value	Meaning
IJBXRETC	IJBXBUFS	Buffers not free. The CLEAR request is accepted and initiated. The user buffers are still in use by the system, and they are not available to the application before the IJBXSECB is posted.

New error codes given back when the CLEAR request is rejected:

Return Code	Value	Meaning
R15	X'08'	Request rejected. Specific error information is provided in the XPCCB field IJBXRETC.

The XPCC error code that may be set in the XPCCB control block at the same time with the return code above:

Field Name	Value	Meaning
IJBXRETC	IJBXAPPC	APPCVM error. The internal APPCVM SENDERR request failed. The error information provided by APPCVM is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the error code IJBXAPPC - has format A:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDERR function code	The APPCVM function which was rejected.

<b>Field Name</b>	<b>Value</b>	<b>Meaning</b>
IJBVMRC	APPCVM IPRCODE value	An error occurred before APPCVM SENDERR was initiated. The VM reason code is stored in IJBVMRC. For explanation see APPCVM documentation.

New reason codes given back when the CLEAR request was accepted and initiated, but its completion failed, or when the CLEAR request is terminated because of path disconnection.

The XPCC reason code in the XPCCB control block has the following format:

Field Name	Value	Meaning
IJBXREAS	IJBXAPRS	APPCVM reason code. The internal APPCVM SENDERR request completed unsuccessfully. The error information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXAPRS +IJBXABDC	APPCVM reason code. The internal APPCVM SENDERR request was terminated by a SEVER request from the communication partner. The APPC path is internally severed. The information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXUEXI	APPCVM reason code. The internal APPCVM SENDERR request was terminated by an unexpected interrupt on this path. The APPC path is internally severed. The information presented by APPCVM with the external interrupt is stored in the receiving user area IJBXRUSR.
IJBXREAS	IJBXAPPV	APPCVM protocol violation. The internal APPCVM SENDERR request was terminated by an APPCVM request from the communication partner violating the supported protocol. The APPC path is internally severed. The information presented by APPCVM with the external interrupt - caused by the unsupported request - is stored in the receiving user area IJBXRUSR.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXAPRS (and IJBXAPRS+IJBXABDC) - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDERR function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP, IJBXVCOD, IJBXVWRC	APPCVM EIB fields: IPTYPE, IPCODE, IPWHATRC	An error or sever occurred when the APPCVM SENDERR completed. The VM error information contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The APPCVM error information in the XPCCB field IJBXRUSR - set with the reason code IJBXUEXI - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDERR function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP	APPCVM EIB fields: IPTYPE	An unexpected interrupt occurred before the APPCVM SENDERR completed. The VM interrupt code contained in the external interrupt data is stored in IJBXVMEI. For explanation see APPCVM documentation.

The APPCVM completion information in the XPCCB field IJBXRUSR - set with the reason code IJBXAPPV - has format B:

Field Name	Value	Meaning
IJBXVMFC	APPCVM SENDERR function code	The APPCVM function for which the interrupt occurs.
IJBXVMEI Subfields: IJBXVTYP, IJBXVCOD, IJBXVWRC	APPCVM EIB fields: IPTYPE, IPCODE, IPWHATRC	An APPCVM SENDERR completed with a protocol violation from the other side. The VM information contained in the external interrupt data is stored in IJBXVMEI. If the external interrupt type is SRI, then IPTYPE is the only data presented. For explanation see APPCVM documentation.

After successful completion of the CLEAR request the application is in SEND state. In case the request fails, the state does not change. However, the state will be 'N', if the path is severed by the partner.

The values may be:

State	When state occurs	XPCC function that may be issued
SEND 'S'	When CLEAR completes successfully - IJBXSECB posted.	SENDER with/without data DISCONN DISCPRG
Link inactive 'N'	After CLEAR completes unsuccessfully because the partner responded with SEVER - IJBXSECB posted.	DISCONN

# Mapping of XPCC Functions to APPC/VM Functions

## CONNECT

An XPCC CONNECT request is internally transformed to an APPCVM CONNECT function.

The APPCVM macro is issued with the following operands depending on the release of the VM host system.

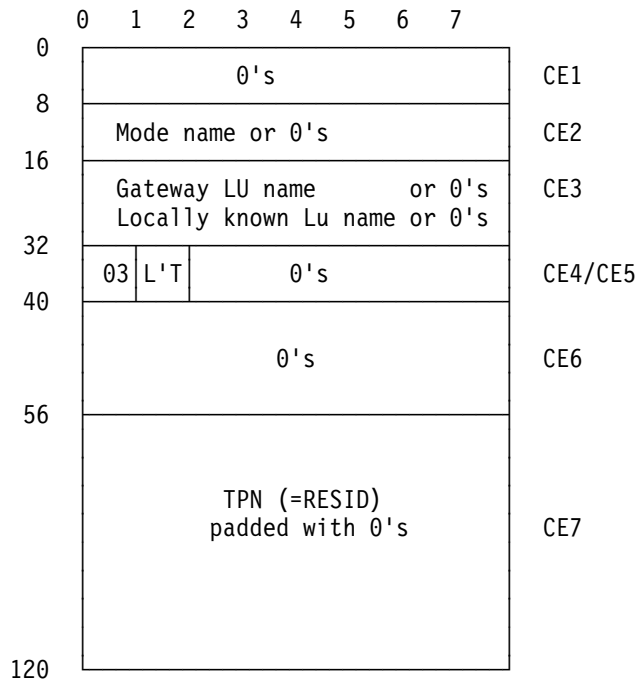
For VM/SP release 5:

```
[name] APPCVM CONNECT,
      PRMLIST=pointer_to_parameter_list,
      CONTROL=NO,
      RESID=pointer_to_SQL/DS_data_base_name,
      SYNCLVL=NONE,
      WAIT=NO
```

For VM/SP release 6, in case VTAM gateway information is supplied:

```
[name] APPCVM CONNECT,
      PRMLIST=pointer_to_parameter_list,
      CONTROL=NO,
      RESID=pointer_to_SQL/DS_data_base_name,
      SYNCLVL=NONE,
      WAIT=NO,
      BUFFER=pointer_to_parameter_list_extension,
      BUFLLEN=pointer_to_length_of_parameter_list_extension,
      FMH5=NO
```

Layout of CONNECT parameter list extension:





- CE1 - Reserved
- CE2 - Mode name of target LU if externally specified
- CE3 - Gateway LU name and target LU name if externally specified
- CE4 - 03 = security(PGM); CP extracts USERID/PASSWD from APPCPASS directory statement
- CE5 - Length of Transaction Program Name has the value 1 to 8
- CE6 - Zero for 03 security(PGM); (CP extracts security information from APPCPASS)
- CE7 - Transaction Program Name is the VM RESID.

### SENDR with Data

An XPCC SENDR request with data to be sent (IJBXBLN=0) is internally transformed to an APPCVM SENDDATA function. The length of the reply buffer may not be zero; it has to be at least four bytes long.

The APPCVM macro is issued with the following operands:

```
[name] APPCVM SENDDATA,
        PRMLIST=pointer_to_parameter_list,
        RECEIVE=YES,
        PATHID=pointer_to_path_identifier,
        BUFLIST=YES,
        BUFFER=pointer_to_send_buffer_address_list,
        BUFLLEN=pointer_to_sum_of_send_buffer_lengths,
        ANSLIST=YES,
        ANSBUF=pointer_to_receive_buffer_address_list,
        ANSLEN=pointer_to_sum_of_receive_buffer_lengths,
        WAIT=NO
```

- The send buffer supplied to XPCC has to contain the APPC logical record length in the first 2 bytes (LL), and the SNA GDS ID (general data stream identifier X'12FF') in the following two bytes.
- The XPCC reply buffer has to be at least 4 bytes long.
- The XPCC reply buffer will contain the original APPC/VM data. The application has to remove LL fields and X'12FF' fields from the APPC logical records.

### SENDR without Data

An XPCC SENDR request without data to be sent (IJBXBLN=0) is internally transformed to an APPCVM RECEIVE function.

The APPCVM macro is issued with the following operands:

```
[name] APPCVM RECEIVE,
        PRMLIST=pointer_to_parameter_list,
        PATHID=pointer_to_path_identifier,
        BUFLIST=YES,
        BUFFER=pointer_to_send_buffer_address_list,
        BUFLLEN=pointer_to_sum_of_send_buffer_lengths,
        WAIT=NO
```

## **CLEAR**

An XPCC CLEAR request is internally transformed to an APPCVM SENDERR function.

The APPCVM macro is issued with the following operands:

```
[name] APPCVM SENDERR,  
      PRMLIST=pointer_to_parameter_list,  
      PATHID=pointer_to_path_identifier,  
      WAIT=NO
```

## **DISCONNect**

An XPCC DISCONNect request is internally transformed to an APPCVM SEVER,TYPE=NORMAL function, if the requesting side is in SEND state 'S'.

The APPCVM macro is issued with the following operands:

```
[name] APPCVM SEVER,  
      PRMLIST=pointer_to_parameter_list,  
      TYPE=NORMAL,  
      PATHID=pointer_to_path_identifier
```

An XPCC DISCONNect request is internally transformed to an APPCVM SEVER,TYPE=ABEND function, if the requesting side is in RECEIVE state 'R'.

The APPCVM macro is issued with the following operands:

```
[name] APPCVM SEVER,  
      PRMLIST=pointer_to_parameter_list,  
      TYPE=ABEND,  
      PATHID=pointer_to_path_identifier
```

## **DISConnect/PuRGe**

An XPCC DISConnect/PuRGe request is internally transformed to an APPCVM SEVER,TYPE=ABEND function.

The APPCVM macro is issued with the following operands:

```
[name] APPCVM SEVER,  
      PRMLIST=pointer_to_parameter_list,  
      TYPE=ABEND,  
      PATHID=pointer_to_path_identifier
```

## VSE XPCCV/APPCVM Sample Protocols as Used by SQL/DS

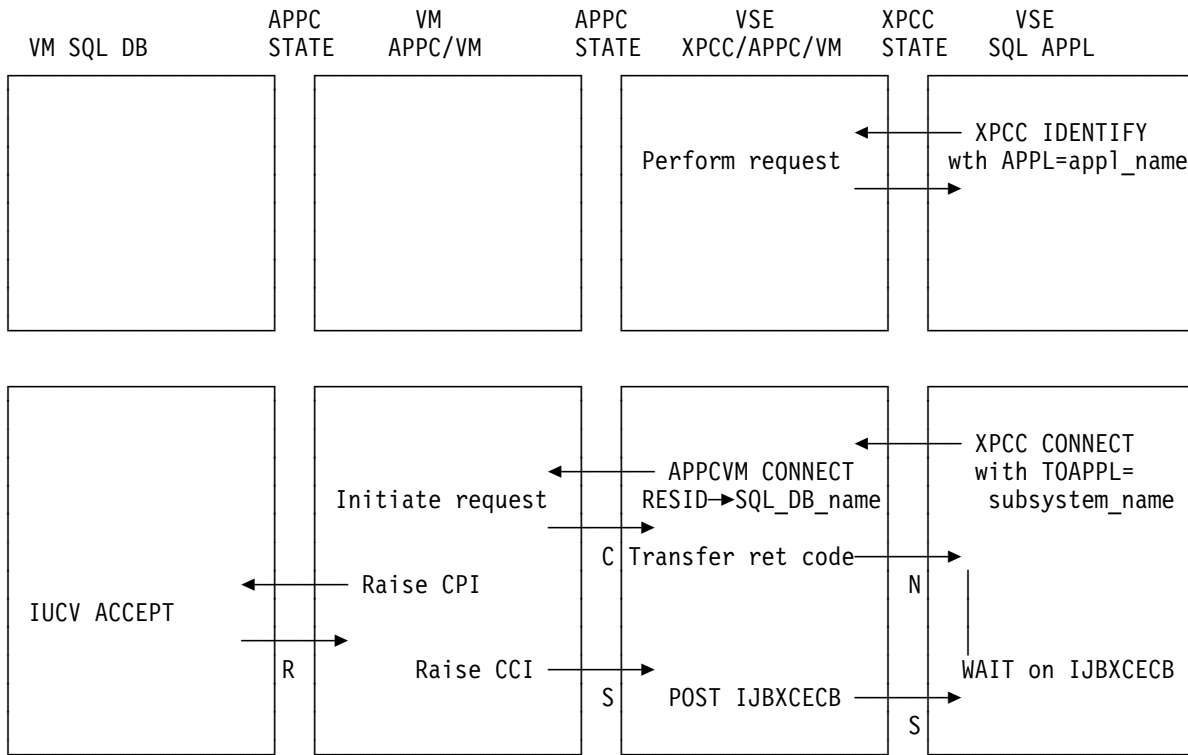


Figure 239. Successful XPCCV CONNECT

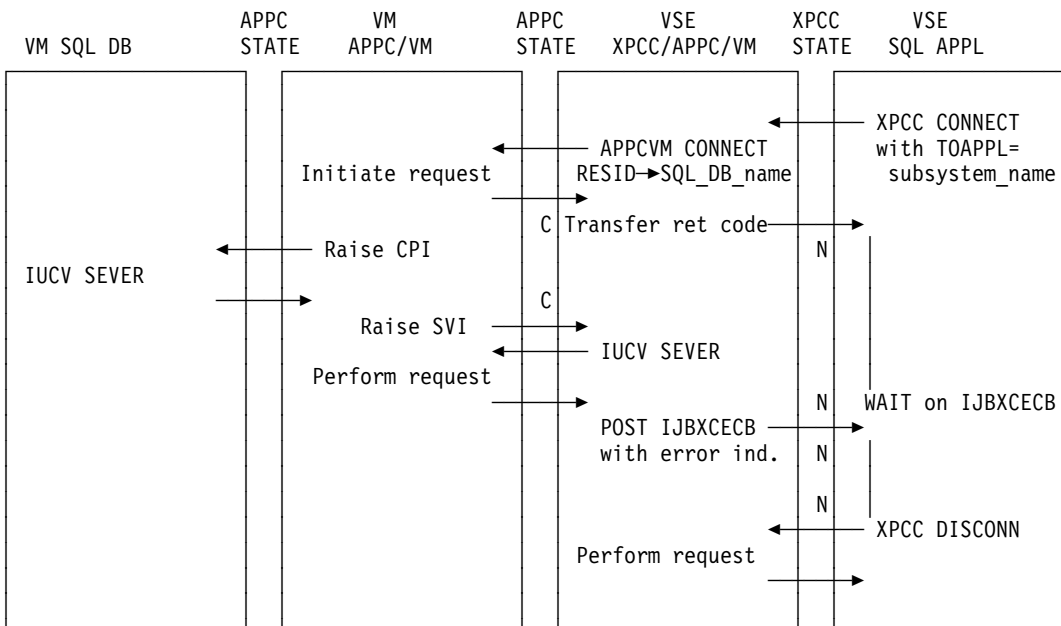


Figure 240. Rejected XPCCV CONNECT





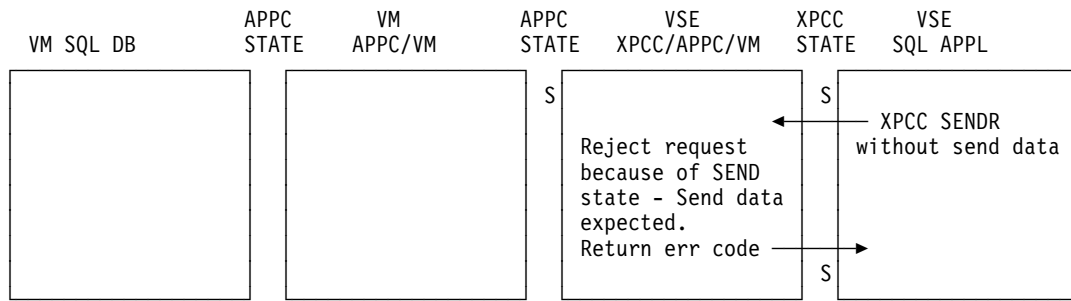


Figure 244. Rejected XPCCV SENDR (Invalid State)

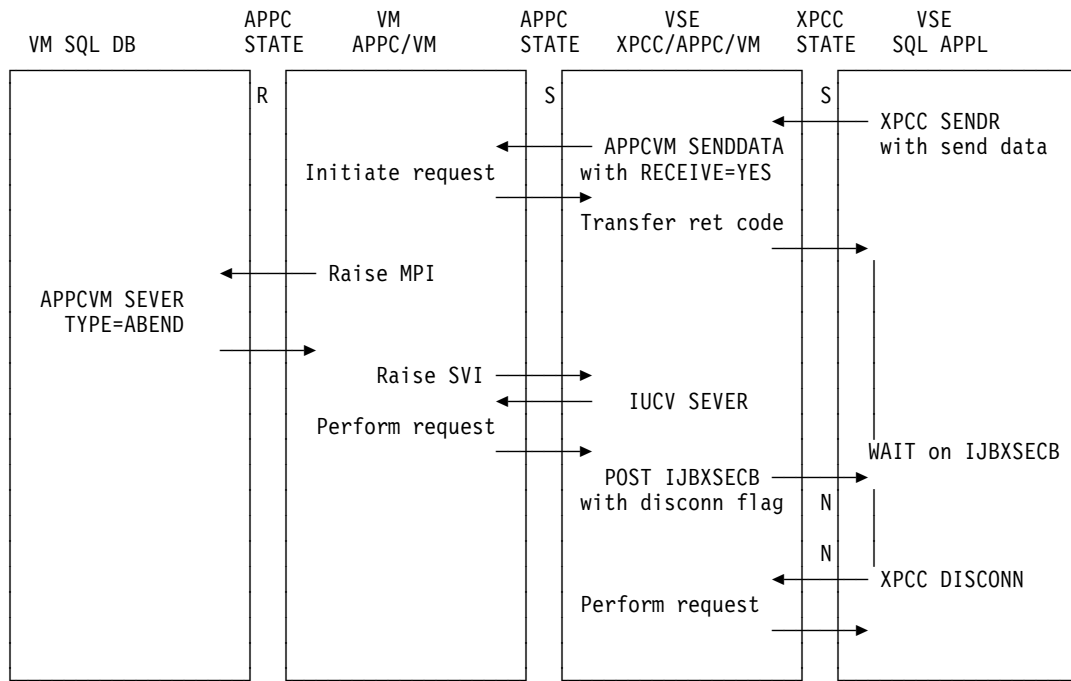


Figure 245. Rejected XPCCV SENDR (Path Disconnected by APPCVM SEVER)

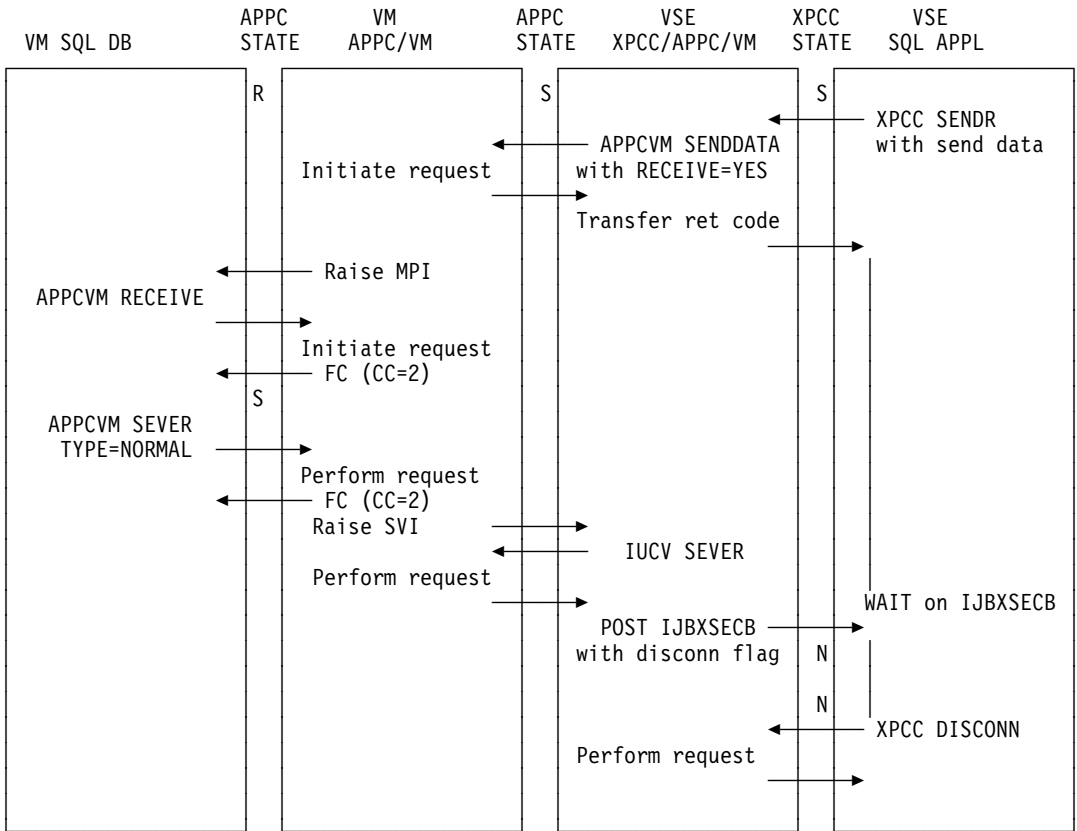


Figure 246. Rejected XPC SENDR (Path Disconnected by APPCVM SEVER)

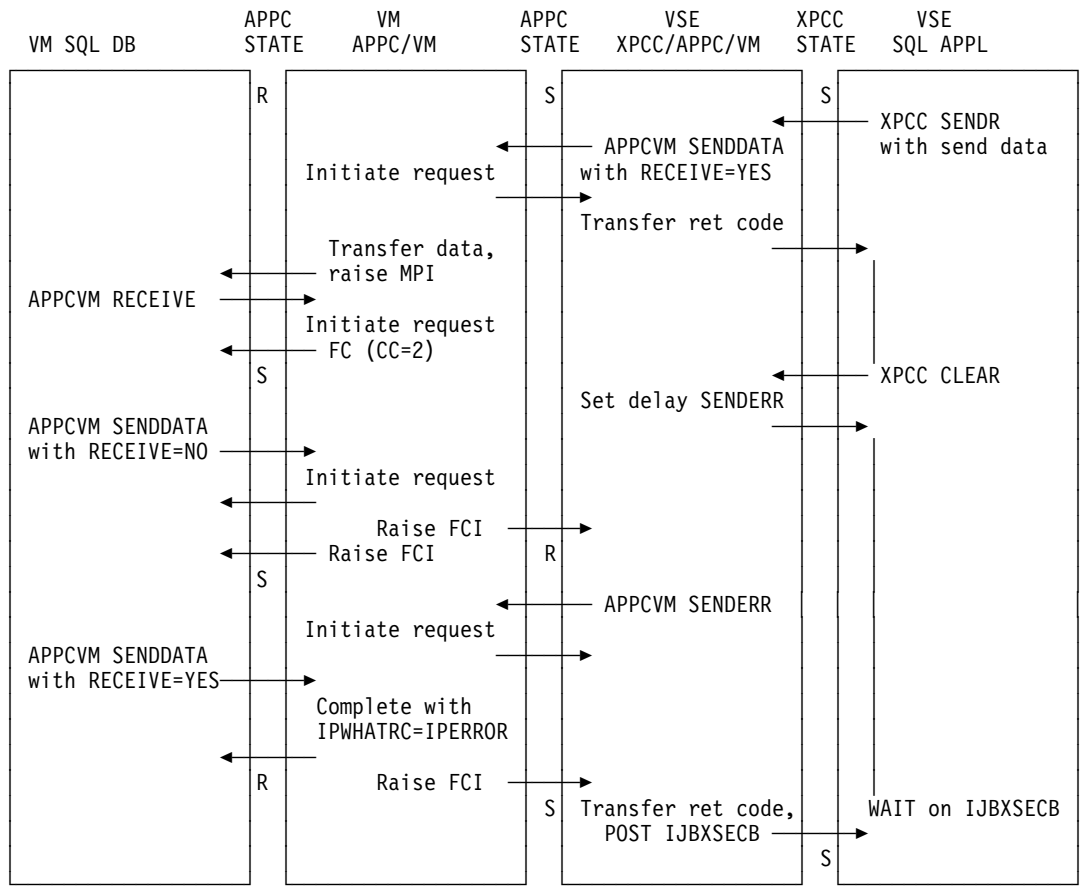


Figure 247. XPC CLEAR (Simulated by APPCVM SENDERR)



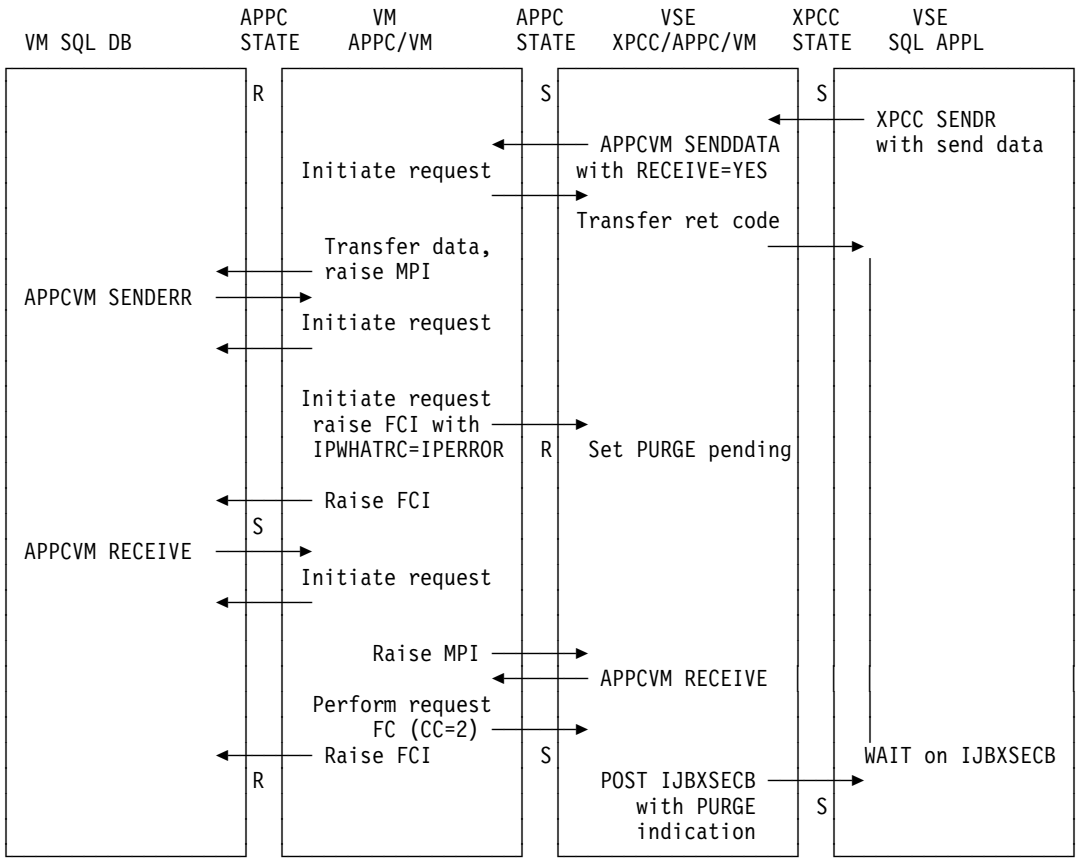


Figure 248. PURGE'd XPC STATE SENDR (Simulated by APPCVM SENDERR from Partner)

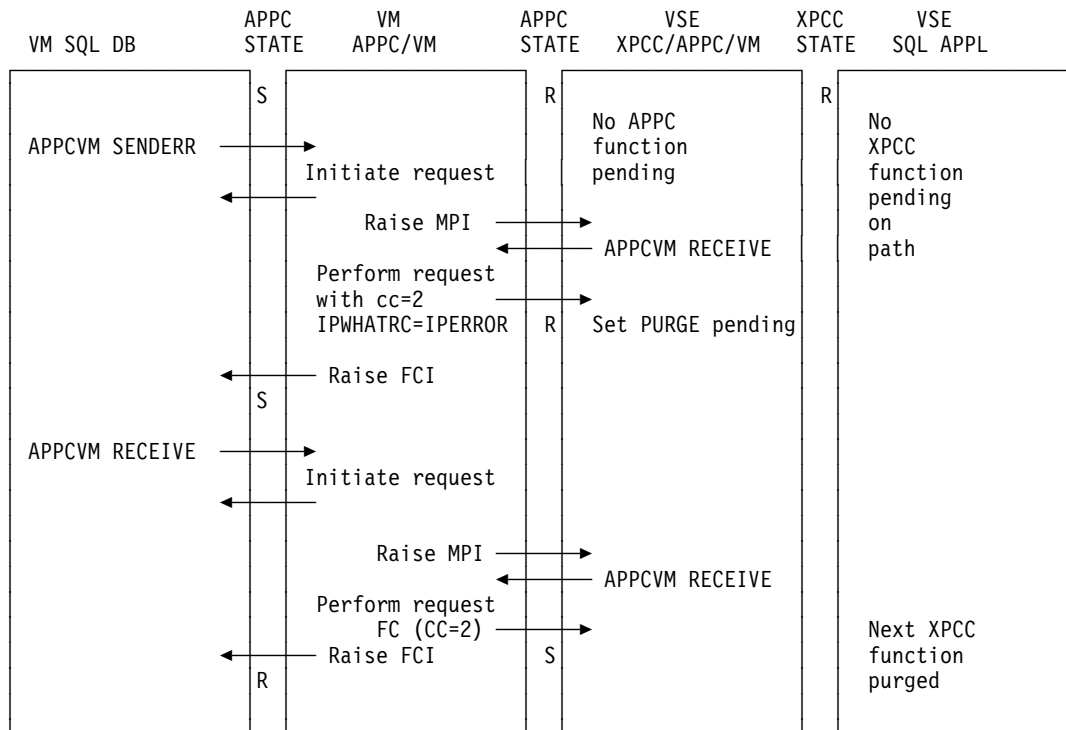


Figure 249. APPCVM SENDERR from Partner (Handled like XPCC PURGE-State R)

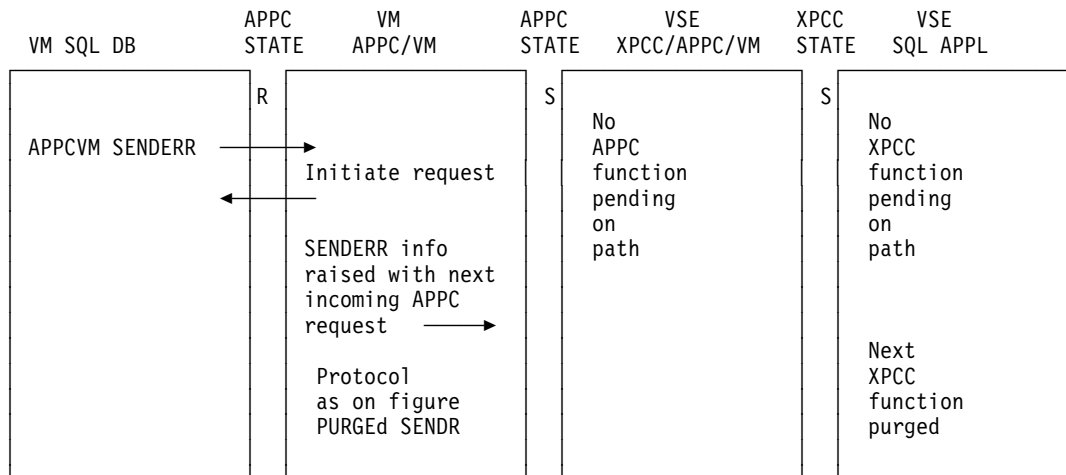


Figure 250. APPCVM SENDERR from Partner (Handled like XPCC PURGE-State S)

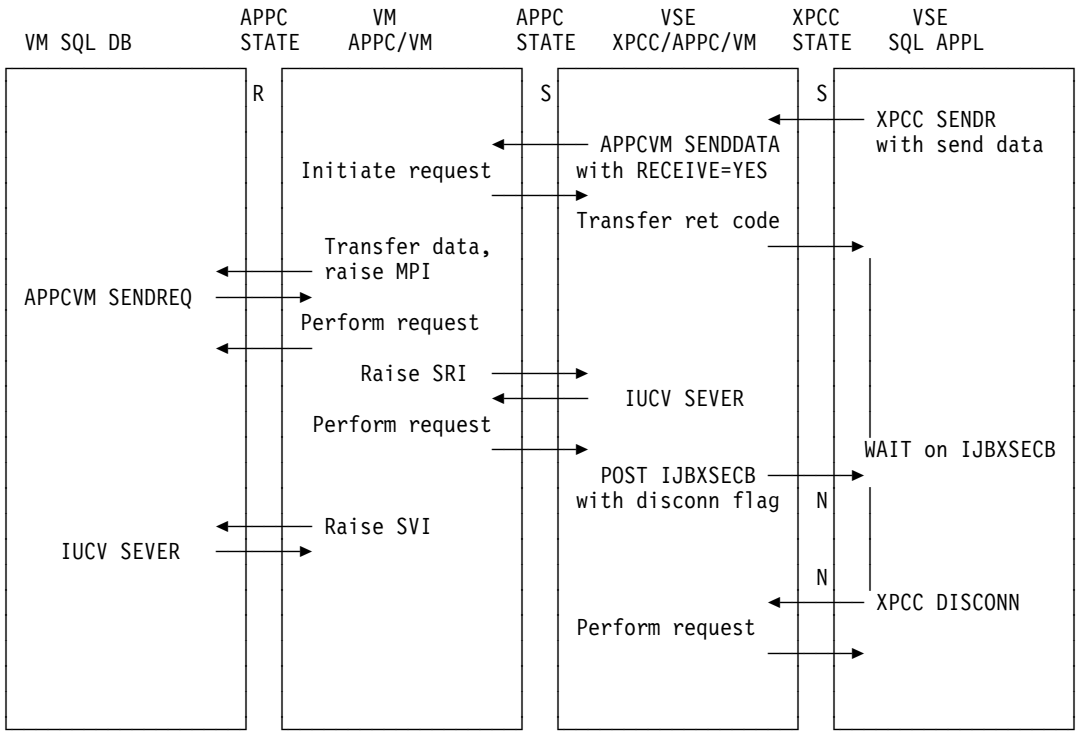


Figure 251. Invalid Response from Partner

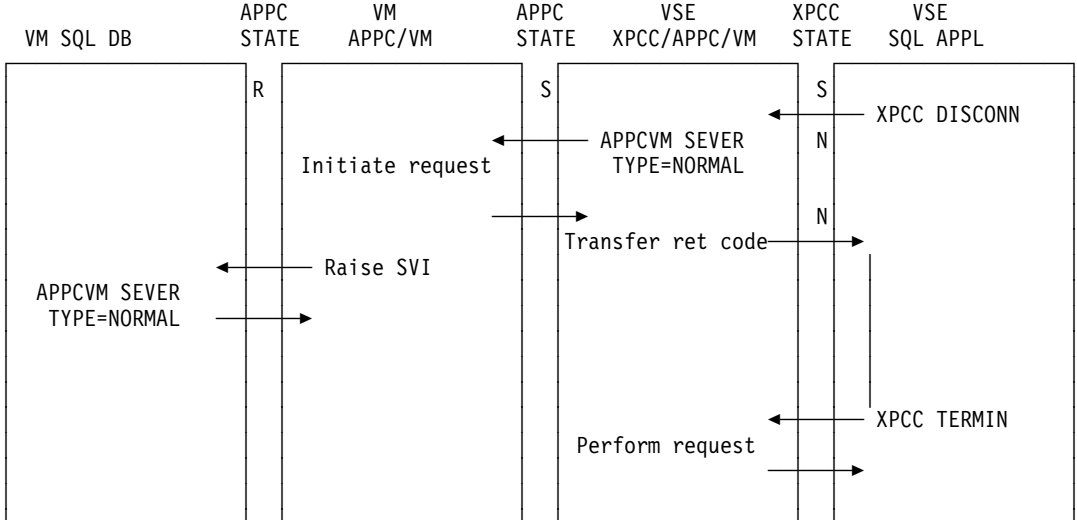


Figure 252. XPC DISCONNECT (APPCVM SEND State)

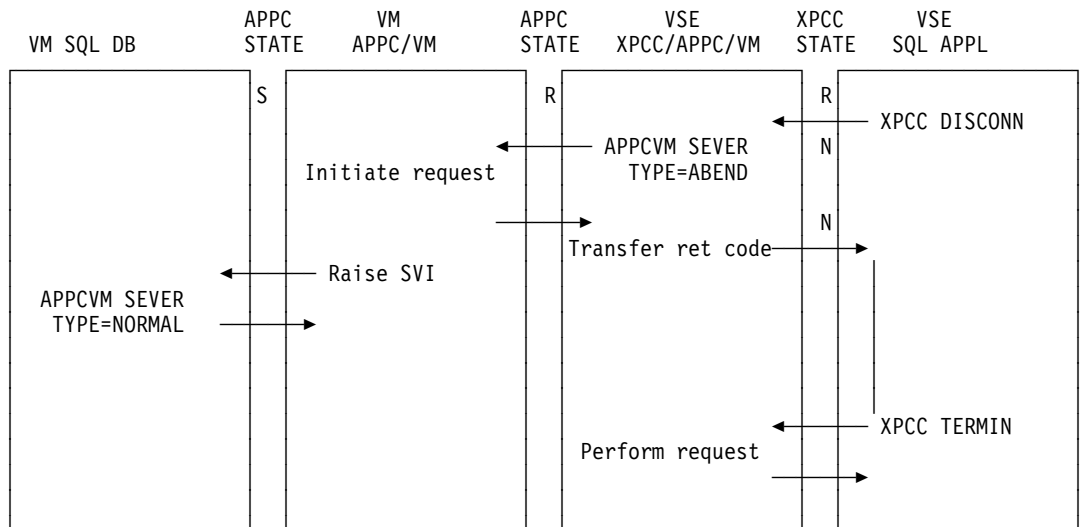


Figure 253. XPC DISCONNECT (APPCVM RECEIVE State)

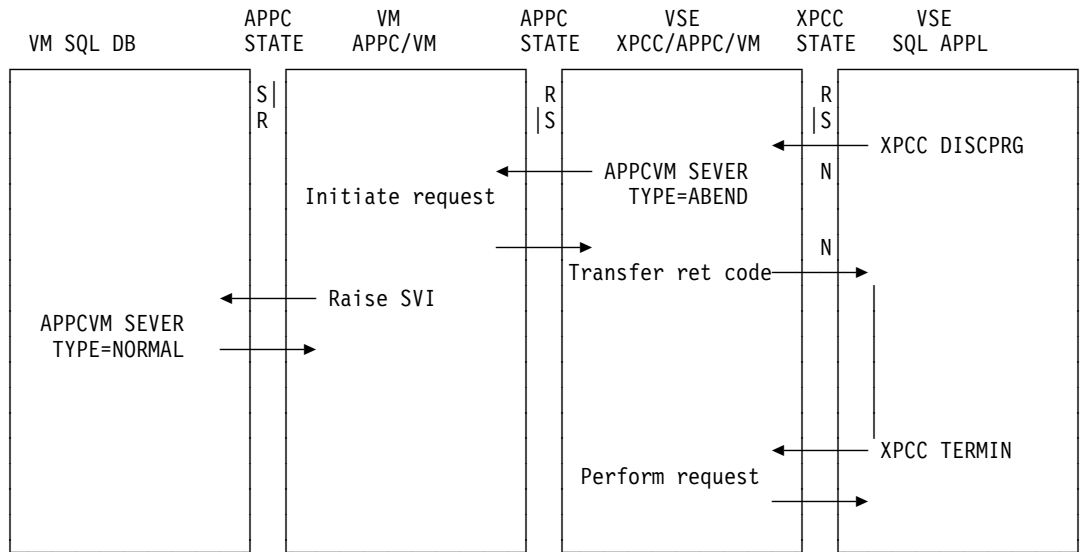


Figure 254. XPC DISCONNECT/PURGE

## Acronyms

### APPC/VM External Interrupt Codes

Interrupt	Meaning	Caused by:
CCI	Connection complete interrupt	Own application issued a CONNECT request with WAIT=NO, and partner ACCEPTs connection.
CPI	Connection pending interrupt	After partner issued a CONNECT request.
FCI	Function complete interrupt	Own application issued a RECEIVE, SENDDATA, or SENDERR request with WAIT=NO, and partner responded appropriately.
MPI	Message pending interrupt	Partner issued a RECEIVE, SENDDATA, or SENDERR request, and own status is R and no function pending.
SRI	SENDREQ interrupt	Partner issued a SENDREQ request.
SVI	SEVER interrupt	When the partner terminates the communication by issuing a SEVER request. May occur when function pending.

## APPC/VM - XPCCV/APPCVM States

APPCVM State	XPCCV/APPCVM State	Meaning
C		Connection pending - intermediate state
	N	Link not active - intermediate state (Connection pending or connection severed)
R	R	RECEIVE state
S	S	SEND state
SV		SEVER state (Set only by FCI because of path severed by partner)

---

## Appendix E. Performance Monitoring Interface

The interface for the VSE Performance Monitor, VSE/PT, used to be the Monitor Call Class 3. This interface had 2 major disadvantages:

1. the MC has a negative performance impact for a VSE Guest on VM, and
2. it was not supported by the VSE Program Check Handler, that means, VSE/PT needed to modify the New PSW for Program Checks to get control.

VSE/PT will not run on VSE/ESA anymore and the Monitor Calls have been removed from the Supervisor Code.

The new performance monitoring interface is described in the following section.

---

### Interactions

- A performance monitor can get control through a Branch Interface, that means, basically the MCs are replaced by Branches.
- The code which will get control via the Branch must reside in the SVA, the address of its entry point must be stored at label PTAIFCOD in area VSEPTCOM (DSECT generated by MAPPTCOM) which is anchored off SYSCOM (label IJBPTCOM).

This code gets control in Supervisor State with interrupts disabled, that means, that code must not generate any interrupt (for example, Page Fault).

If it does, the system goes into a Hard Wait.

This code must be PFIxed.

Control is transferred in Primary Space Mode.

The Addressing Mode, in which the monitor gets control, can be defined by the monitor's code itself:

- If the high-order bit (bit 0) in field PTAIFCOD is set (=1) then control is transferred in AMODE(31),
- else control is passed in AMODE(24).

Upon return to the supervisor the task must be in the same state (registers(except Reg. 15), PSW, etc.) as it was at time of entry.

The AMODE must be reset to what it was before control was given to the monitor.

- The following sequence of steps must be implemented by a performance monitor for initialization:
  - A performance monitor must identify itself to VSE via  
SUBSID NOTIFY,NAME=<VSPT>,
  - The same task then must load the interface code into the system GETVIS (PFIxed) and store the entry point address of that code into field PTAIFCOD.
- Thereafter, any task of the performance monitor can activate monitoring.
  - A monitoring period is initiated via  
MODFLD FIELD=VSPTFLG,NEWVAL=(r1)or<LABEL>  
if the label notation is used, LABEL must identify a field containing  
X'00000001'

if register notation is used, the register must contain the value X'00000001'.

- The task which issues the MODFLD must have identified itself to VSE via SUBSID NOTIFY,NAME=<VSPT>, before.

From now on, control will be passed from the various points in the supervisor to the performance monitor code as described above.

- Thereafter, any task of the performance monitor can deactivate monitoring.
  - A monitoring period is deactivated via MODFLD FIELD=VSPTFLG,NEWVAL=(r1)or<LABEL> if the label notation is used, LABEL must identify a field containing X'00000000' if register notation is used, the register must contain the value X'00000000'.
  - The task which issues the MODFLD must have identified itself to VSE via SUBSID NOTIFY,NAME=<VSPT>, before.
- Finally a performance monitor must inform VSE about its termination via SUBSID REMOVE,NAME=<VSPT>.
- With the branch also a function code is passed which allows the performance monitor to decide which monitor point has passed control. In addition, following registers are used as interface between the supervisor and a performance monitor when control is passed:

R14 = A(2-byte function code)

R15 = Entry point address (that means, base register for the code of the performance monitor)

BR 2(R14) allows return from the performance monitor to supervisor

There are 17 function codes defined at the moment.

- FC=1;  
replaces MC(4). Event: I/O interrupt
- FC=2;  
replaces MC(8). Event: Start I/O
- FC=3;  
replaces MC(12). Event: System goes into allbound state
- FC=4;  
replaces MC(16). Event: SVC interrupt (not for SVC 107)
- FC=5;  
replaces MC(20). Event: I/O request put into channel queue
- FC=6;  
replaces MC(24). Event: Process set time request
- FC=7;  
replaces MC(28). Event: Interrupt from clock comparator
- FC=8;  
replaces MC(32). Event: Fetch for LTA phase
- FC=9;  
replaces MC(36). Event: Free LTA
- FC=10;  
new. Event: Program check interrupt



- FC=11;  
new. Event: External interrupt
  - FC=12;  
new. Event: Phase loaded from SVA
  - FC=13;  
new. Event: Phase loaded from a library
  - FC=14;  
replaces MC(44). Event: Interval timer interrupt
  - FC=15;  
new. Event: Set CPU timer (caused by user task)
  - FC=16;  
new. Event: Set CPU timer (caused by external interrupt)
  - FC=17;  
new. Event: GETVIS updates counter of used pages
  - FC=18;  
new. Event: Phase not found
- VSEPTCOM is moved above the 64KB line in supervisor. It is extended by several fullwords.

PTASVCTB DC A(SVCTAB)

PTACTAB DC A(CHNTAB)

PTCPYTOT DC F(# of totally available copy blocks), set by IPL

PTCPYAVL DC F(# of currently free copy blocks), set by IPL and modified by CCW translation

PTCPYHWM DC F(# of so far unused copy blocks), set by IPL and modified by CCW translation

PTAIFCOD DC A(Entry point of monitor for supervisor)

PTAINITT DC A(field INITTIME)

PTACMSAV DC A(field COMMSAV)

PTARID DC A(field VSEPTRID)

PTAPTSA DC A(field VSEPTSA)

PTAIDCB DC A(anchor of chain of XPCC ID control blocks)

Field PTAPFT has been removed.

- A new macro, SGVSEPT, handles the branch interface to the performance monitor, it must be coded in the supervisor wherever control is to be passed. The interface is  
SGVSEPT PROBE=,SAVE=  
POBE is a mandatory parameter, it defines the function code passed to the monitor,  
SAVE is optional, the default is YES; it indicates whether the caller wants registers 14 and 15 saved.



---

# Index

## Numerics

64-bit page frames 220

## A

Access Register  
    Access Register Translation 312  
Access Register Save Area 451  
Access Register Translation 307, 312  
Access Registers 307  
    Access Register Translation 307  
    Primary-List Bit 308  
    Save Area 451  
Access-List Entries 308  
Access-Register (AR) Mode 312  
ACCTCOMN (Job Accounting Table) 436  
Address Spaces 317  
Address spaces, size of 192  
address table  
    partition control block 445  
ALD (Access-List Designation) 310  
ALESERV Macro 321  
ALESN (Access-List-Entry Sequence Number) 308  
ALET 307  
anchor table 285  
Anchor table (ANCHTAB) 285  
appendixes 433  
area  
    ERBLOC 499  
    supervisor patch 401  
ART 307  
Assign  
    Stored Assignment Table entry 476  
ASTE (ASN-Second-Table-Entry) 310  
AT CVT 461  
attention  
    task 348  
automatic volume recognition (AVR) 103  
AVR (automatic volume recognition) 103

## B

B-tree (index set) 335  
balanced group 66  
BLKTBE (VIO Block Table Entry) 460  
block protection 98  
blocked page-out 227  
blocked paging  
    concepts 226  
    page-in 193, 226, 237, 238, 239  
    pre-page-out 193, 226, 227, 228  
    tuning considerations for page-out 235

blocked paging (*continued*)  
    unconditional page-out 193, 226, 231, 233  
bound state  
    table 59  
bound state setting  
    I/O 118  
    resource 125

## C

cancel  
    codes 409  
    routine 76  
CAW (Channel Address Word) 467  
CCB  
    copy blocks 161, 162  
    special notes 491  
CCB (command control block) 488  
CCH (channel check handler) 370  
CCW  
    copy block 161, 163, 165  
CCWTCB (translation control block) 161  
channel  
    address word (CAW) 467  
    check entry  
        device 503  
    check handler (CCH) 370  
    check handling 370  
    check severity detect routine 370, 372  
    Control Table (CHNTAB) 485  
    queue table (CHANQ) 97, 486  
    status word (CSW) 468  
channel program  
    copying 168  
    translation 161, 168  
CHANQ (Channel queue table) 97, 486  
CHNTAB (Channel Control Table) 485  
CLASSADR (dynamic class table entry) 440  
CLIMADR (Dynamic Class and System Limits Table) 439  
CMT 329  
code-fix area, temporary 401  
command control block (CCB) 488  
Companion Manuals x  
COMREG (partition communications region) 446  
control block  
    Command (CCB) 488  
    ECB 454  
    IORB 493  
    partition 444  
    partition (PCB) 446  
    PIB 446

- control block (*continued*)
  - PIB2 446
  - RCB 455
  - SAACOMM 77
  - SCB 444
  - system (PCB) 446
  - task 447
- control blocks 45
  - CPU 46
  - page 0 45
  - Ported from OS/390 461
  - prefix page 45
  - task selection 46
  - TD address table 46
  - TD communication region 46
  - TD CPU control block 46
- Control Unit Initiated Reconfiguration 105
- control word
  - channel (CAW) 467
- copy blocks
  - CCB 162
  - CCW 163, 165
- CPCBADR (dynamic class control block) 440, 441
- CPU activation 45
- CPU control blocks 46
- CPU initialization 45
- CPU management 45
  - activation 45
  - initialization 45
  - termination 45
- CPU termination 45
- Cross Memory Communication
- CRW (channel report word) 369
- CRW handling
- CSW (channel status word) 468
- CUIR 105
- CVT
  - OS/390 461
  - VTAM 461

## D

- DASD sharing
  - FETCH 133, 360
  - lock management 129
  - second level directory - SLD 360
- Data Space 193, 307
- Data Spaces
  - compared to address spaces 319
  - Control Block Structure 322
  - Introduction To 318
- data structures
  - PGQSYS 238
  - POSL 233
  - PREPGENT 233
  - PREPGOTB 233

- Data-Only Space
  - definition of 319
  - illustration of 319
- DCT (device characteristics table) 105
- Debug commands
  - Debug ON 424
  - Debug Select 427
  - Debug Show 428
  - Debug Stop 429
  - Debug Trace 425
- Debugging Facilities
  - Features 411
    - Address Compare Stop 413
    - Event tracing 413
  - How to find it 414
  - Locate command 430
  - Overview 411
  - SHOW command 432
  - Trace Areas 415
  - Trace Entries 416
    - All Register 417
    - Cancel 421
    - Data 422
    - Dispatcher Exit 418
    - External Interrupt 420
    - I/O Interrupt 419
    - MC Debug entries 423
    - Program Check 416
    - Start I/O 419
    - Supervisor Call 421
    - Switch 422
    - Task Selection 420
    - User Data 423
- DEVCB (device control block) 210
- device
  - characteristics table (DCT) 105
  - control block (DEVCB) 210
  - scheduling 97
  - Usage Counters (DVCUSCNT) 483
- DEVTAB 349
- DIB (Disk Information Block Table) 496
- DIBX (disk information block extension table) 498
- directory
  - entry 344
  - list support 344
  - search 348
- disk
  - information block
    - (DIB) table, CKD 496
    - (DIB) table, FBA 498
    - extension (DIBX) table 498
- Disk Error Recovery 113
- dispatcher 47
  - partition cleanup 88
  - partition preparation 88
  - system dump interfaces 77

dispatcher (*continued*)  
 VSE/ICCF support 86  
 DPDTAB (Page Data Set Table) 208  
 DSPSERV Macro 320  
 DSRCHNx (first level chain control blocks) 349  
 DU-AL 313  
 DUCT (Dispatchable-Unit Control Table) 308  
 DVCUSCNT (Device Usage Counters) 483  
 dynamic class 32, 440  
 CLASSADR 440  
 control block (CPCBADR) 441  
 CPCBADR 440  
 information 440  
 limits table (CLIMADR) 439  
 table entry (CLASSADR) 440  
 dynamic class table 32, 440  
 dynamic partition 31  
 dynamic space GETVIS area 31  
 DYNCLASS ID=LOAD 440

**E**

ECB (Event Control Block) 454  
 ERBLOC Area 499  
 ERP Message Writer 115  
 ERPIB (error recovery procedure information block) 370, 511  
 error  
 bytes (fixed storage) 406  
 entries 110, 111  
 entry  
 device 501  
 hard wait codes 403, 406  
 I/O  
 Block (ERBLOC) 499  
 processing  
 I/O 110  
 Recovery  
 Disk 113  
 Message Writer 115  
 Tape 114  
 recovery entry  
 device 502  
 recovery procedure information block (ERPIB) 370, 511  
 ESA 307  
 ESA/390 307  
 Event Control Block (ECB) 454  
 exponential average  
 for partition deactivation 251  
 extended addressability  
 introduction to 318  
 Extended Page Address (EPA) 192  
 Extended Page Number (EPA#) 192  
 extent  
 information entry 477

external interrupt 27

## F

FCEPGOUT request 248  
 FCHWORK area 350  
 fetch  
 control flow 333  
 DASD sharing 360  
 second level directory 360  
 service task 361  
 FETCH/LOAD processing 348  
 first level chain control blocks (DSRCHNx) 349  
 first level interrupt handler 21  
 VSE/ICCF support 86  
 fix information block 162, 168  
 fixed storage  
 error bytes 406  
 fixed storage locations 400  
 fixing  
 pages  
 permanently (PFI) 244  
 temporarily (TFI) 243  
 FLPTR (Free List Pointer) 472  
 Free List Pointer (FLPTR) 472  
 FREEREAL requests 247  
 FRPL (FETCH request parameter list) 356

## G

gates  
 permanently closed 58  
 permanently opened 58  
 switchable 58  
 type and value 59  
 gating mechanism  
 internal 55  
 GETPRTY 67  
 GETREAL request 242  
 GETVIS  
 area  
 control information 263  
 Control information 278  
 PFI processing 274  
 subpool 266

## H

hard wait error codes 403, 406  
 high priority dispatching 49  
 Home-Space Mode 312

## I

I/O  
 error processing 110  
 interrupt 21

- I/O (*continued*)
  - interrupt handler 101
  - request enqueuer 97
  - table interrelationship 469
- I/O device
  - channel check entry 503
  - error entry 501
  - error recovery entry 502
- I/O Request Block (IORB) 493
- ICCF pseudo partition 49
- IDAL (indirect data address list) 162
- IDAL blocks 166
- IDAWs (indirect data address words) 166
- identification
  - of a partition 94
- II
  - VM/VSE Interface Routines 386
- indirect data address
  - list (IDAL) 162
  - words (IDAWs) 166
- information blocks
  - fix 168
- Input/Output Request Block (IORB) 493
- interfaces
  - partition cleanup 88
  - partition preparation 88
  - system dump 77
  - VSE/ICCF 86
- interrupt
  - external 27
  - first level interrupt handler 21
  - handler 101
  - I/O 21
  - machine check 27
  - missing 116
  - page fault 23
  - processors 21
  - program check 22
  - second level interrupt handler 21
  - supervisor call (SVC) 27
- interrupt process
  - clock comparator 44
  - I/O 44
  - SVC 42
- introduction to extended addressability 318
- Invalid Page Frame Queue (IPFQ) 202
- INVPAGE requests 248
- IORB (Input/Output Request Block) 493
- IORE 211, 228, 231, 238, 239
- IPFQ (Invalid Page Frame Queue) 202, 204, 205, 216, 219, 223, 224, 225, 237, 244
- IPFQ64 (Invalid Page Frame Queue) 216, 220

## J

- job accounting
  - Device SIO accounting 483
  - general information 375
  - initialization
    - tables 375
  - updating account information 376
  - updating SIO counters 376
  - user interface 377
- job accounting partition table (ACCTABLE) 436
- job accounting table (ACCTCOMN) 436

## L

- LB (library block) 335
- LDT (Library Definition Table) 349
- library
  - sublibrary 335
  - virtual 342
- library (NLIB) 335
- library block (LB) 335
- library offset table (LOT) 349
- Linkage Stack 325
- load point
  - phase 356
- Locate command 430
- LOCK
  - dead lock detection (logic) 125
  - function 117
  - logic 118, 120
  - macro 117
- lock file 129
  - block capacity 131
  - CPU N flag 130
  - data blocks 130, 131
  - entry 130
  - format 131
  - header 129
  - record id 130
- lock management 117
  - control blocks 453
  - DASD sharing 129
  - flags 123
  - internals 122
  - lock options 119
  - return codes 124
  - SVC 110 117
- logical transient
  - area occupancy and activity 452
- Logical Transient Area (LTA) 1
- Logical Transient Key (LTK) 96
- Logical Transient Owner (LTID) 96
- logical transient save area 451
- Logical Unit Block
  - Extension table 475

Logical Unit Block (*continued*)  
 table (LUBTAB) 473  
 LOT (Library Offset Table) 349  
 LRA special operation exception handling 225  
 LTA  
 occupancy and activity 452  
 save area 451  
 LTA (Logical Transient Area) 1  
 LTID (Logical Transient Owner) 96  
 LTK (Logical Transient Key) 96  
 LUB  
 Extension table 475  
 LUBTAB (Logical Unit Block table) 473

## M

machine check  
 analysis 367  
 handling 367  
 recording 367  
 machine check interrupt 27  
 macros  
 ALESERV 321  
 DSPSERV 320  
 FETCH 344  
 GENL 344  
 LOAD 344  
 LOCK 117  
 UNLOCK 117  
 UNLOCK ALL 121  
 UNLOCK ALL,JC=EOJ 121  
 UNLOCK SYSTEM=sys-id 121  
 MCAR 367  
 missing interrupt handler 116  
 MODE=ESA 307

## N

NLIB (library) 335  
 NPARTS  
 number of partitions 93  
 number of partitions  
 NPARTS 93

## O

OS/390 461  
 OS/390 ECB (OS/390 Event Control Block) 454  
 OS/390 Event Control Block (ECB) 454  
 OS/390 Extended ECB (OS/390 Extended Event Control Block) 454  
 OS/390 Extended Event Control Block (ECBE) 454

## P

page 0 45

page data set (PDS) 189  
 Page Data Set Table (DPDTAB) 208  
 page fault  
 handler 23  
 handling overlap 240  
 interrupt 23  
 pseudo 24  
 page fault processing 215  
 page frame 189  
 table entry (PFTE) 203  
 table/(PFT) 202  
 page frames above 2GB 220  
 page handling routines 221  
 page I/O request element (IORE) 211  
 page management 189  
 Data Space 193  
 Page management tables  
 device control block (DEVCB) 210  
 initialization of PTE 201  
 IORE 211  
 page data set table (DPDTAB) 208  
 page frame table (PFT) 202  
 page frame table (PFTE) 202  
 page I/O request element (IORE) 211  
 page table 200  
 Page table assignment string (PTAS) 206  
 page table entry (PTE) 200  
 Page to disk assignment string (PDAS) 206  
 PAGEINTB 207  
 PGINENT 207  
 PISL 206, 237, 238  
 POSL 206, 227  
 PREPGENT 207  
 PREPGOTB 207  
 segment table entry 199  
 Page Manager Address Space (PMRAS) 196  
 Page Selection Queue (PSQ) 202  
 page table 200  
 Page table assignment string (PTAS) 206  
 page table entry (PTE) 200  
 Page to disk assignment string (PDAS) 206  
 page-in 238, 239  
 queue entry 222  
 request 224, 246  
 Page-In queue entry 222  
 page-out 231, 233  
 page-out queue entry 222  
 page-table 197  
 Page-table assignment string (PTAS) 197  
 PAGEINTB 207  
 partition  
 communications region (COMREG) 446  
 control block (PCB) 446  
 control block address table (PCBATAB) 445  
 control block interrelationship 444  
 dynamic 31

partition (*continued*)  
   dynamic class control block (CPCB) 441  
   identification 94  
   information block  
     (PIB) 446  
     extension (PIB2) 446  
   key definitions 93  
   static 30  
 partition balancing 66, 68  
 partition cleanup 88  
 Partition Identification Key (PIK) 94  
 Partition Identifier (PID) 94  
 partition preparation 88  
 PASN-AL 313  
 patch areas 401  
 PCB (partition control block) 446  
 PCBATAB (PCB address table) 445  
 PDAS (Page to disk assignment string) 206  
 Performance Monitoring 555  
 PFIX request 244  
 PFREE request 247  
 PFT (Page Frame Table) 202  
 PFTE (Page Frame Table Entry) 203  
 PGINENT 207  
 PGQI (page-in queue) 215, 222, 223, 225  
 PGQO (page-out queue) 222, 224, 228, 229, 230, 232  
 PGQSYS (system queue) 224, 232, 234  
 PGQSYS(system queue) 223, 231, 234, 237, 238, 239  
 phase  
   relocatable 356  
 Phase Load Trace Table 402  
 physical  
   Input/Output Control System (PIOCS) 97  
   Physical Transient Area (PTA) 1  
   Physical Unit Block  
     extension (PUBX) 480  
     extension area 479  
     Ownership Table (PUBOWNER) 483  
     PUB2 482  
     table (PUBTAB) 477  
   PIB (partition information block) 446  
   PIB2 (partition information block extension) 446  
   PID (Partition Identifier) 94  
   PIK (Partition Identification Key) 94  
   PIOCS (Physical Input/Output Control System) 97  
   PISL 206, 237, 238, 239  
   POSL 197, 206, 227, 231, 233  
   POST Routine 55, 62, 63  
   prefix page 45  
   PREPGENT 207, 233  
   PREPGOTB 207, 233  
   Primary-List Bit 308  
   Primary-Space Mode 312  
   problem program (PP) save area 449  
   procedures  
     pgotunig 236  
   processing of interrupts 21  
   program  
     retrieval 331  
   Program Call (PC-ss)  
   program check  
     handling of a normal 22  
     interrupt 22  
   PRTY command 66  
   pseudo page fault 24, 241  
   PSL (Page State List)  
     PISL 206, 237, 238, 239  
     POSL 206, 227, 231, 233  
   PSQ (Page Selection Queue) 202, 204, 205, 216, 219,  
     224, 225, 244  
   PSQ64 (Page Selection Queue) 216, 220  
   PTA (Physical Transient Area) 1  
   PTAS (Page table assignment string) 206  
   PTE (page table entry) 200  
   PUB 477  
     extension area 471  
     Ownership Table (PUBOWNER) 483  
     PUB2 482  
   PUB2 482  
   PUBOWNER (PUB Ownership Table) 483  
   PUBTAB (Physical Unit Block Table) 477  
   PUBX  
     accessing a PUBX entry 479  
   PUBX (Physical Unit Block Extension) 480  
   PUBXAREA (PUB Extension Area) 471, 479

## R

RAS  
   monitor 372  
   transient 372  
 RAS Linkage Area (RASLINK) 508  
 RAS Monitor Table (RASTAB) 509  
 RAS Transient Area (RTA) 1  
 RASLINK (RAS Linkage Area) 508  
 RASTAB (RAS Monitor Table) 509  
 RCB (Resource Control Block) 455  
 Re-IPL 379  
   II Console 385  
 recorder file  
   table 504  
 reentry rate 251  
 relocatable phase 356  
 relocation dictionary read request (RLD) 356  
 RELPAG request 248  
 Request queues  
   PGQSYS 238  
 Resident Tape Error Recovery 114  
 resource  
   descriptor entry 57  
   descriptor table addressing 56



Resource Control Block (RCB) 455  
 RFTABLE 504  
 RID (Routine Identifiers) 26  
 RLD (relocation dictionary read request) 356  
 RLD read request 359  
 RMODE 342  
 routine  
   POST 55, 62, 63  
   RPOST 62  
   UNPOST 62, 63  
   WAIT 55  
 Routine Identifiers (RID) 26  
 RPOST Routine 62  
 RTA (RAS Transient Area) 1

## S

SAACOMM 77  
 save areas 449  
   LTA 451  
     problem program 449  
     system 451  
 SCB (space control block) 444  
 scheduler 100  
   device 100  
 SDL (system directory list) 342  
 SDT (Sublibrary Definition Table) 349  
 second level interrupt handler 21  
 Secondary-Space Mode 312  
 segment table 197, 199  
 selection 216  
   pool 216  
   pool queues 216  
 SETPRTY 67  
 shared virtual area (SVA) 342  
 SHOW command 432  
 SIGP 45  
 SMCB (Storage Management Control Block) 261  
 space control block (SCB) 199, 444  
 static partition 30  
 status flags  
   task 59  
 status word  
   channel 468  
 storage  
   allocation  
     dynamic 263  
     dynamic partition support 259  
     page manager tables 260  
     segment table handling 260  
   management 261  
   management control block (SMCB) 261  
   protection key 93  
 Storage Protection Override  
 stored assignment table entry 476

subpool  
   chain table entry 281  
   GETVIS 266  
   index table entry 281  
 Subsystem Storage Protection  
 supervisor  
   areas allocated by IPL 13  
   call interrupt 27  
   patch areas 401  
 SVA (shared virtual area) 342  
 SVC  
   110 (X'6E') 117  
   (LOCK) 117  
   (UNLOCK) 117  
   65 (X'41' - CDDELETE) 284  
   65 (X'41' - CDLOAD) 283  
   call description 27  
   interrupt 27  
 SVC 43 (DYNCLASS) 88  
 SYSDEF 45  
 SYSFIL processing 99  
 SYSLIB sublibrary 342  
 SYSRES 342  
 system  
   directory list (SDL) 342  
   limits table (CLIMADR) 439  
   save area 451  
 system console attention processing 109  
 system dump interfaces 77  
 system files 99  
   FBA 99  
 system resource owner 49  
 system task  
   activation 369, 371  
   LCK 129  
   RAS 369, 371, 372  
   selection 29  
 system/partition 29

## T

table  
   Channel Control Table 485  
   CHANQ 486  
   CLASSADR 440  
   CLIMADR 439  
   COMREG 446  
   CPCBADR 440, 441  
   DIB 496  
   DIBX 498  
   LUB Extension 475  
   LUBTAB 473  
   of Bound States 59  
   PCBATAB 445  
   PUB 477  
   PUB2 482

- table (*continued*)
  - PUBOWNER 483
  - PUBX 480
  - RFTABLE 504
  - SRQTAB addressing 56
  - Stored Assignment Information 476
  - THTAB 512
  - TIBATAB 50
- tape fetch
- task
  - and partition key definitions 93
  - cancel
  - cancel exit routine 76
  - identification 94
  - information block (TIB) 211
  - status flags 55, 59
  - termination 73
- task selection 48
- task selection control blocks 46
- task termination 73
- tasking service
  - DYNCLASS - SVC 43 88
- TCPU 46
- TD address table 46
- TD communication region 46
- TD CPU control block 46
- TDATAB 46
- TDCOMREG 46
- Teleprocessing Balancing (TPBAL) 255
- temporary code-fix area 401
- TFIX request 243
- TFREE request 247
- THTAB (Track Hold Table) 512
- TIB (Task Information Block)
  - IORE 228, 231, 238, 239
- TIB Address Table (TIBATAB) 50
- TIBATAB (TIB Address Table) 50
- time slicing 66
- TP Balancing 255
- track hold processing 98
  - sample 513
- Track Hold Table (THTAB) 512
- track protection 98
- translation control block (CCWTCB) 161
- TSC 46, 47
  - TSCT queue 48
- TSCC 47, 48
- TSCP 47, 48
- TSCT 47, 48
- TSCT queue 48
- tuning for page-out 235
- turbo dispatcher
  - design 37
  - dispatcher 47
  - Example 38
  - IPL 37

- turbo dispatcher (*continued*)
  - ready-to-run queue 48
  - virtual storage layout 41
- TXT read request 356, 359

## U

- unconditional page-out 231, 233
- UNLOCK
  - function 117
  - macro 117
- UNPOST routine 62, 63
- user task
  - selection 29

## V

- VCT (volume characteristics)
  - table 105
  - update 105
- VIO Block Table Entry (BLKTBE) 460
- VIO Communication Area (VIOCM) 457
- VIO File Identification Entry (VIOTABE) 459
- VIO POINT request 248
- VIO Table Entry (VTABE) 457
- VIOCM (VIO Communication Area) 457
- VIOTABE (VIO File Identification Entry) 459
- virtual library 342
- virtual storage layout 41
- VM
  - VM/VSE Interface Routines 386
- volume characteristics table (VCT) 105
- VSE/ICCF
  - dispatcher support 86
  - first level interrupt handler support 86
- VTABE (VIO Table Entry) 457
- VTAM Vector Table Address (AT CVT) 461

## W

- wait queue 55, 57
- WAIT Routine 55

## X

- XMOVE 317
  - Introduction To Data Spaces 318
- XPCC/APPCVM Protocol 519
  - overview 519
  - VSE XPCC/APPCVM Sample Protocols as used by SQL/DS 543

## Z

- z/Architecture 307





Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

XXXX-XXXX-XX



DSMBEG323I STARTING PASS 2 OF 4.

DSMBEG323I STARTING PASS 3 OF 4.

DSMBEG323I STARTING PASS 4 OF 4.

+++EDF258W Cross references were not resolved. Check cross-reference listing to find problems. (Page 568 File: FB8SUM21 SCRIPT)

DSMMOM397I '.EDF#END' WAS IMBEDDED AT LINE 188 OF 'EDFPRF40'