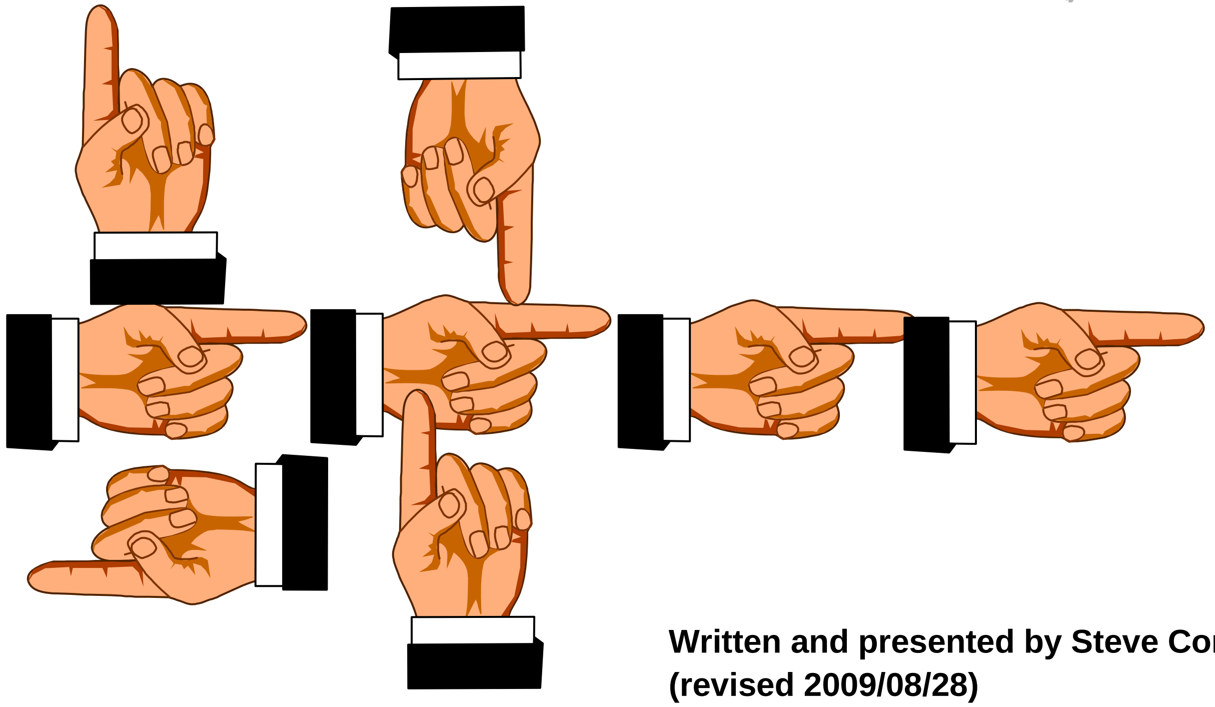
z/OS Control Blocks for Beginners

August 27, 2009
SHARE Session 2238
Denver, Colorado



Written and presented by Steve Comstock for The Trainer's Friend, Inc.
(revised 2009/08/28)

Agenda

- z/OS as a Resource Manager
- Control blocks - definition
- Addresses and pointers
- Chains and anchors
- Control blocks - Address Spaces (ASCB), Tasks (TCB), Requests (PRB, SVRB), Contents supervision (CDE, XTLST)
- Where are the layouts
- Reading the layouts
- The CVT and PSA
- Some details of the above control blocks
- Finding the data: DDLIST, SYSUDUMP, control block chasing
- The study of control blocks and data areas

z/OS Is a Resource Manager

- "Resources" includes
  - I/O Devices and the external hardware configuration
  - Memory and CPs (physical and virtual)
  - Units and Volumes
  - Catalogs, VTOCs, Labels, Data sets, Directories, Members
  - Address spaces
  - Tasks
  - Jobs
  - Memory contents (*e.g.*: programs currently loaded)
  - Users (potential and currently logged on)

z/OS Is a Resource Manager, 2

- To "Manage" resources means
  - To know the location and status of each resource
  - To be able to create, modify, and delete resources

- z/OS does this through the concept of "control blocks" (CB's)

Control Blocks

- A control block is a contiguous string of bytes in memory that represents / describes a resource
- There is one control block for each specific instance of each resource

Control Blocks, 2

- Control blocks are the main mechanism to preserve, change, and communicate the state and status of resources

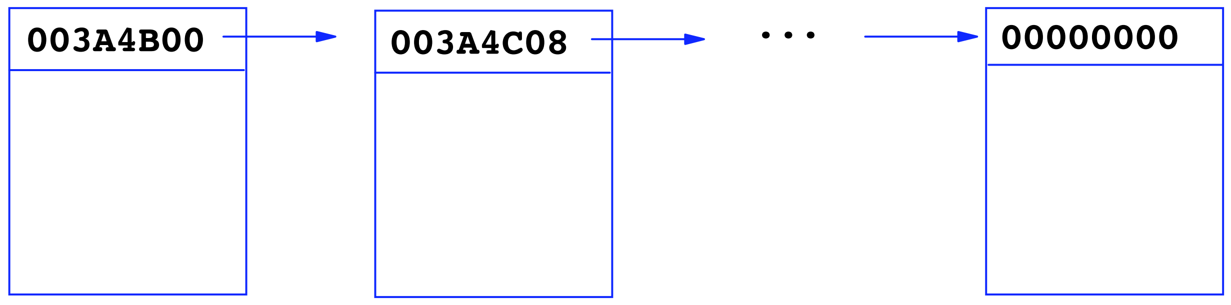
- If you understand control blocks in z/OS, you can design, code, and debug applications in a way that works with the system

Addresses and Pointers

- Although you may well be familiar with these terms, understanding them is integral to grasping how control blocks are used
- ADDRESS - an integer indicating a particular byte in memory
  - Each byte of memory is numbered, starting with zero
  - Addresses in z Series may be 24 bits, 31 bits, or 64 bits long
- POINTER - an area in memory that contains an address: 3 bytes, 4 bytes, or 8 bytes long

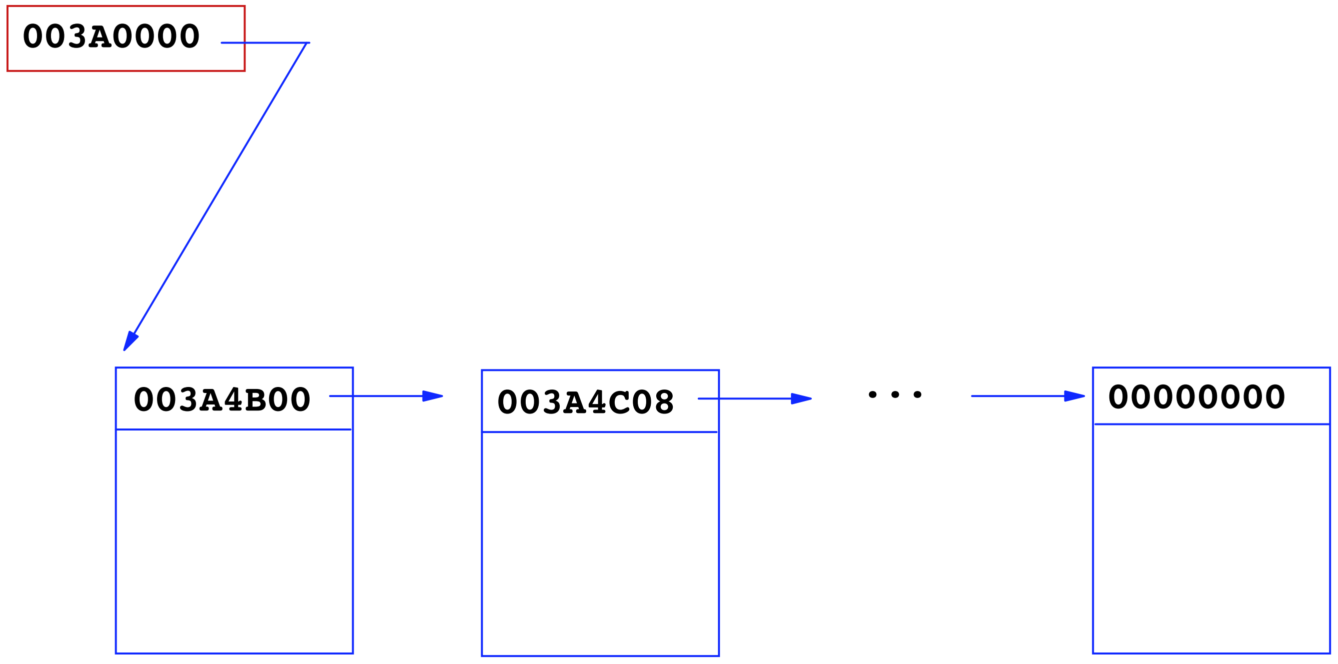
Addresses and Pointers, 2

- Related control blocks are typically connected through pointers: each pointer containing the address of some related control block
  - Special case: a value of all zeros in a pointer is taken to mean "not a valid address" - the null pointer
- Visually, you will often see related ("chained") control blocks represented this way:



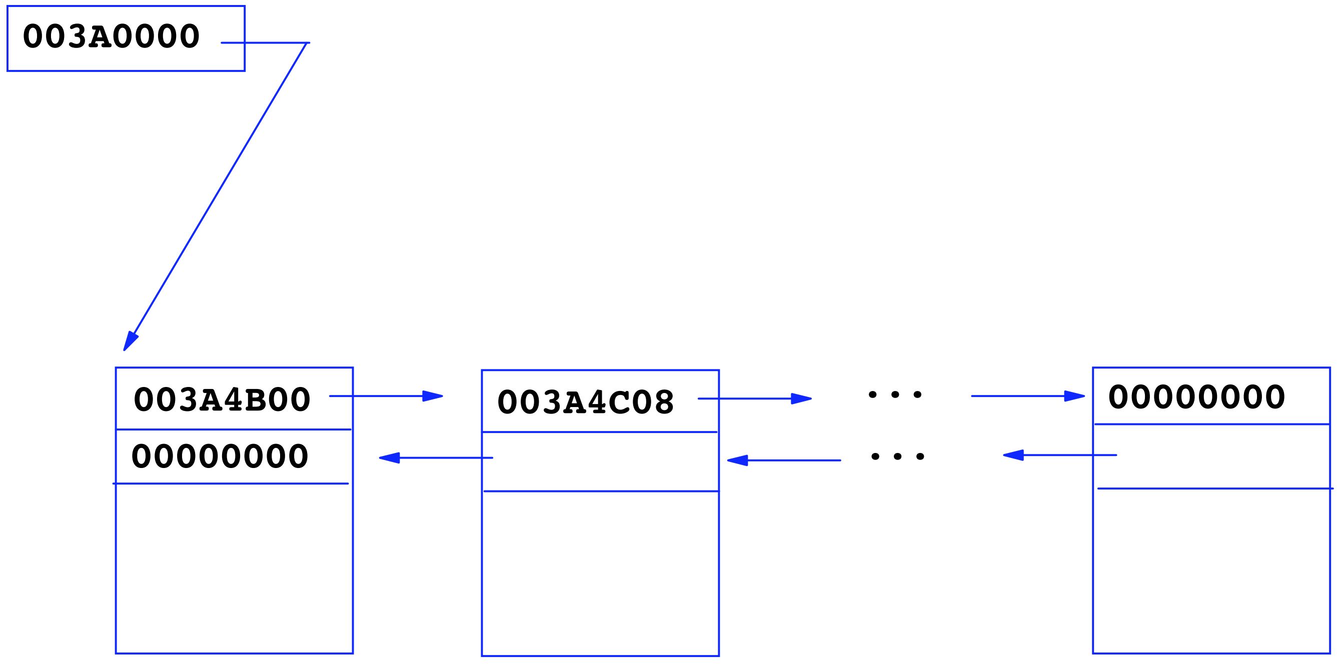
Chains and Anchors

► When control blocks are chained together, the start of the chain is usually kept in a known location, called the <u>anchor</u> (starting point) of the chain



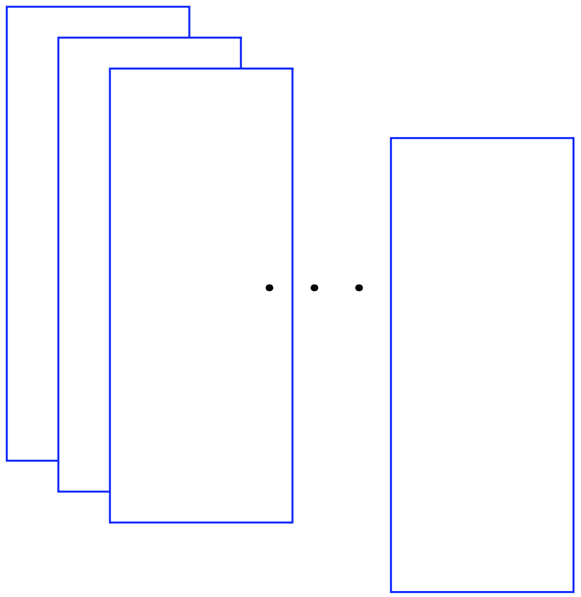
Chains and Anchors, 2

- Sometimes control blocks are chained together in a double linked list, so you can easily move forward or backwards along the chain:



Control Blocks: Address Spaces

- In z/OS, every batch job, every TSO user, and every system task runs in its own address space
  - A virtual storage with addresses from 0 to the maximum, totally unique from every other virtual storage
  - Each address space shares a common copy of z/OS plus the program(s) this job, user, or system task is running



Each address space is assigned a unique integer: its Address Space ID (ASID)

Control Blocks: Address Space Control Blocks

- Each address space is represented by an Address Space Control Block (ASCB)
  - ASCB's are chained together in a double-linked list
  - Some of the other information in an ASCB:

* The ASID

* Pointer to the Address Space Extension Control Block (ASXB)

* Pointer to Region Control Task (RCT) Task Control Block (TCB)

* Lots more

Control Blocks: Tasks

- Work in an address space is organized into **tasks**
  - Each task is generally independent of any other task in the address space (although there are mechanisms for coordination and communication)

- Each task is made up of **requests**
  - Request for a system service
  - Request to run a program

  - Each task is represented by a Task Control Block (TCB) and each request is represented by a Request Block (RB: SVRB, PRB, and some others)

Control Blocks: Tasks, continued

- When an address space is created, z/OS first creates a task to manage the address space - the Region Control Task (RCT)

- The RCT builds tables necessary to support the new address space's virtual storage then attaches 2 subtasks:

  - Dump Task - waits for an abend to occur in the address space

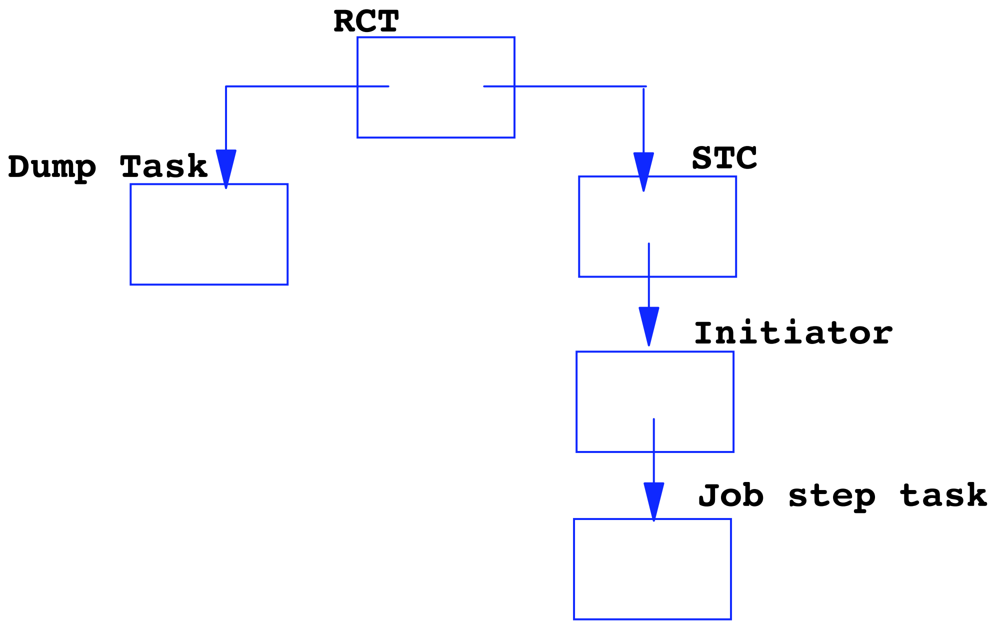
  - Started Task Control (STC) - attaches a task to run the requested program (for batch, the initiator)

Control Blocks: Tasks, continued

- The initiator takes each step in your job, one at a time, allocates the resources the program in the step will need, and ATTACHes the program as the Job Step Task - this is the task that your program will be running under
  - ATTACH is a system service to create a task that is a subtask of the requesting task

Control Blocks: TCBs

- Each task is represented by a Task Control Block (TCB)
- Any task may attach additional subtasks, and TCB's are linked together (mother, daughter, sister tasks), so you might see:

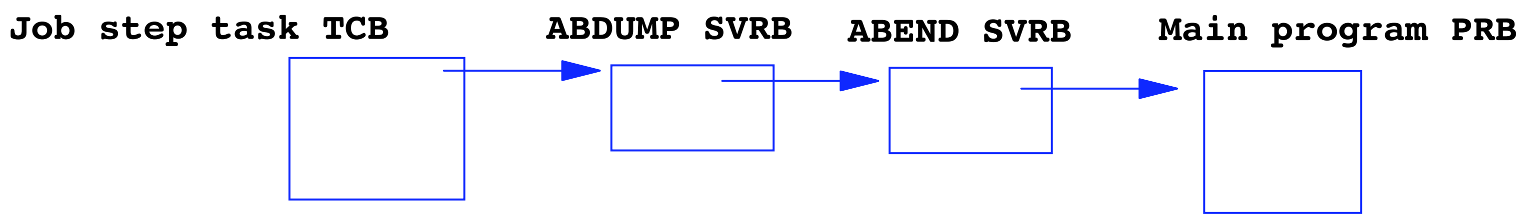


Control Blocks: Request Blocks

- Each TCB has a chain of request blocks (RBs) that represent services and programs to run
  - For example, your application program has a PRB (Program Request Block) to manage the program's status
  - If your program requests a system service such as, say, OPEN, the service will run under an SVRB (SuperVisor Request Block)
  - The first RB off the TCB chain represents the program or service currently running
  - Later RBs represent requestors of the service or program...

Control Blocks: Request Blocks, 2

- Here's the RB chain you are likely to see in a dump from an ABEND:



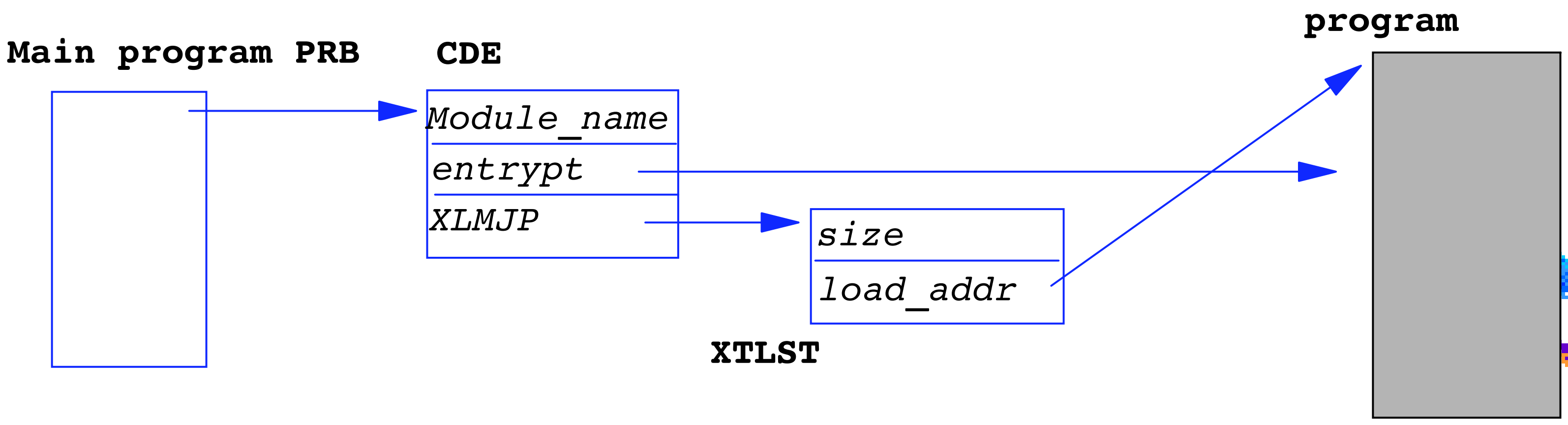
In this situation, your program, running under a PRB, "requested" the Abend service (perhaps not voluntarily)

ABEND saw you had a SYSUDUMP DD statement for the step so it requested the Abdump service to print a dump

ABDUMP is the request running at the time the dump is produced

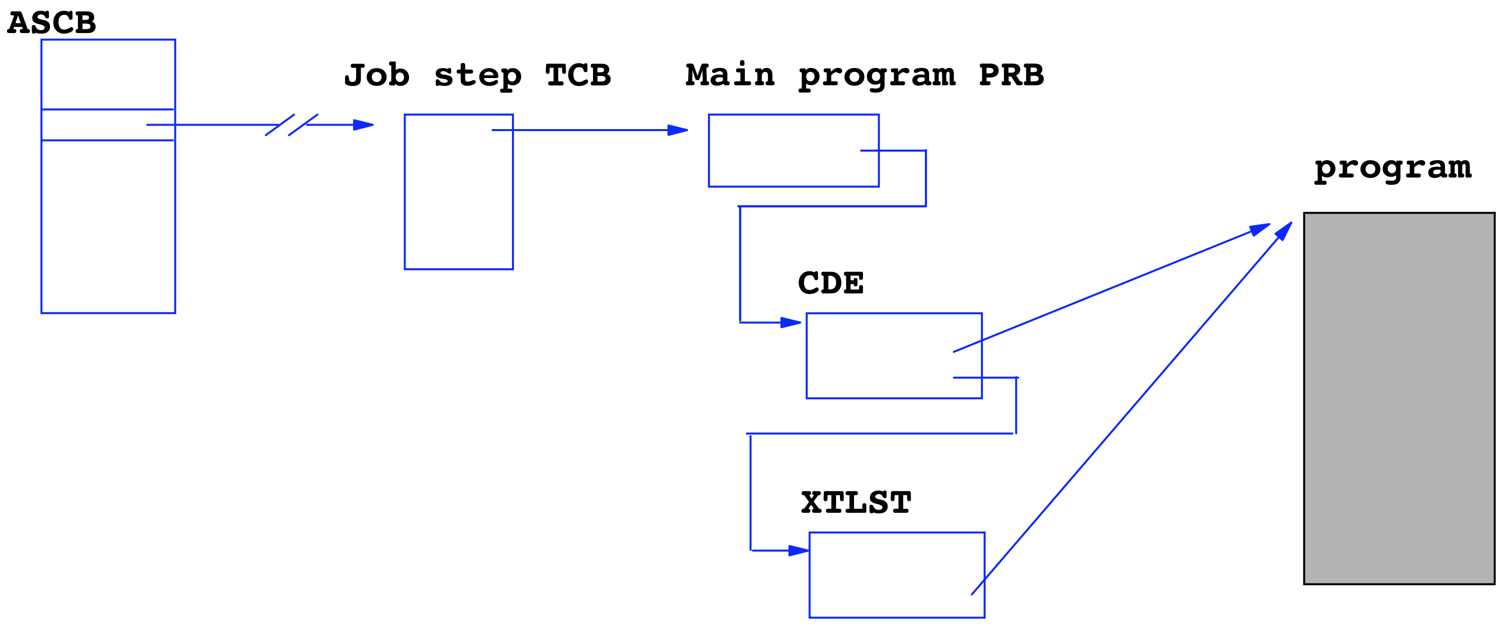
Control Blocks: CDEs and XTLSTs

- But wait! There's [lots] more; for example
  - Your PRB will point to a control block called the CDE (Contents Directory Entry) which contains the module name and entry point address and a pointer to another control block:
    - The XTLST (eXTentLiST) which specifies where the module is loaded and how large it is, so:



Control Blocks: Essential Control Blocks

- So when you are simply running a batch job, here are the control blocks that are normally in play (simple version):



Control Blocks - Where Are the Layouts?

- There are a number of sources of information about z/OS control blocks:

  - The six volume set of "MVS Data Areas" ("Data Area" is a generalization of "Control Blocks"); on the Web at:

http://www.ibm.com/systems/z/os/zos/bkserv/

  - Select "z/OS Elements and Features", "Book", *latest_version*
  - Under the elements and features bookshelf choice, select "List books"
  - Do a Find for "data areas"; read or download

Control Blocks - Where Are the Layouts, 2

- There are a number of sources of information about z/OS control blocks, continued:
  - The common macro libraries on your system contain mapping macros for control blocks
    - SYS1.MACLIB, for example, contains these mapping macros (and more):
      * IKJTCB - maps TCB
      * IHARB - maps all RBs (PRB, SVRB, IRB, etc.)

Control Blocks - Where Are the Layouts, 3

- Different components and products use different libraries; some other examples:
  - CEE.SCEEMAC, for example, contains mapping macros for LE (Language Environment) control blocks
  - DSN*vrr*.SDSNMACS contain mapping macros for various DB2 control blocks
- Some control blocks don't have mapping macros published

Control Blocks - Reading the Layouts

- Control block layouts in macro libraries are hard to read
  - Assembler and, sometimes, some PL/X
  - Sometimes commented mysteriously or misleadingly

- Control block layouts in the Data Areas docs are somewhat easier to read, but still complicated

  - Note also that different products will have control blocks documented in their publication set

Control Blocks - Reading the Layouts, 2

- Here's part of the TCB description from the Data Areas manual:

Offsets					
Dec	**Hex**	**Type/Value**	**Len**	**Name (Dim)**	**Description**
0	(0)	DBL WORD	8	(0)	
0	(0)	X'20'	0	TCB	"*" - TCBPTR
0	(0)	ADDRESS	4	TCBRBP	- ADDRESS OF THE RB FOR EXECUTING PROGRAM. THIS OFFSET FIXED BY ARCHITECTURE.
4	(4)	ADDRESS	4	TCBPIE	- Address of current PIE/EPIE. This field may be tested for zero to determine that there currently is no SPIE/ESPIE exit established for this task Ownership: RTM Serialization: Local Lock
8	(8)	ADDRESS	4	TCBDEB	- ADDRESS OF THE DEB QUEUE
12	(C)	ADDRESS	4	TCBTIO	- ADDRESS OF THE TASK I/O TABLE (TIOT)
16	(10)	BITSTRING	4	TCBCMP (0)	- TASK COMPLETION CODE AND INDICATORS
16	(10)	BITSTRING	1	TCBCMPF	- INDICATOR FLAGS
		1...		TCBCREQ	"X'80'" - A DUMP HAS BEEN REQUESTED
		.1..		TCBCSTEP	"X'40'" - A STEP ABEND HAS BEEN REQUESTED

  - Note the bit string breakdown

Control Blocks - Reading the Layouts, 3

► Here's part of the RB description from the Data Areas manual (it includes PRB, TIRB, SIRB, SVRB in one place):

```
   Offsets
Dec        Hex      Type/Value    Len     Name (Dim)   Description
. . .
192        (C0)     CHARACTER      12                      - FIRST 12 BYTES OF RBSCBX
204        (CC)     ADDRESS         4         RBSXPARM     - 31-BIT PARAMETER LIST ADDRESS
208        (D0)     SIGNED          4         SVRBEND (0) - END OF SVRB (BOTH) (MDC338)
12          (C)     ADDRESS         4         RBCDE (0)    - SAME AS RBCDE1 BELOW
12          (C)     BITSTRING       1         RBCDFLGS     - CONTROL FLAGS
                     1... ....                RBNOCELL     "BIT0" - EXIT SHOULD FREEMAIN THIS SVRB RATHER
                                                             THAN FREECELL MDC008
                     .1.. ....                RBRSV009     "BIT1,,C'X'" - RESERVED
                     ..1. ....                RBCDATCH     "BIT2" - CONTENTS SUPERVISION HAS BEEN ENTERED
                                                            VIA ATTACH ICB444
                     ...1 ....                RBCDSAVE     "BIT3" - EXIT WILL LOAD REGISTERS FROM PRB ON
                                                            RETURN FROM SYNCH TO ROUTINE (MDC345)
                     .... 1...                RBCDNODE     "BIT4" - NO DE SAVE AREA REQUIRED ICB444
                     .... .1..                RBCDSYNC     "BIT5" - SYNCH MACRO INSTRUCTION REQUESTED
                     .... ..1.                RBCDXCTL     "BIT6" - XCTL MACRO INSTRUCTION REQUESTED
                     .... ...1                RBCDLOAD     "BIT7" - LOAD MACRO INSTRUCTION REQUESTED
13          (D)     ADDRESS         3         RBCDE1       - ADDRESS OF CDE, ADDRESS OF LPDE OR ZERO (SEE
                                                           COMMENTS FOR BIT RBTRSVRB)
. . .
```

► Note the sudden change of offset; this occurs on the 11th page of the write up; if you don't look far enough you could miss the fact that PRB + 12 contains the address of the CDE

Control Blocks - Stability

- IBM has long documented the major control blocks pretty thoroughly

- But as the operating system changes, some changes may need to be made to control blocks

- Generally, IBM documents some fields as "Programming Interface Information" (sometimes "General Use Programming Interface" - GUPI)
  - This designation tells the programmer these fields can be relied upon to maintain their functionality; other fields may change their usage

Control Blocks - Stability, 2

- Sometimes a control block has had to be enlarged, to contain more information; there are two approaches used when this happens:
  - <u>Extensions</u> - one field is changed to be a pointer that will point to an extension of the control block
  - <u>Prefix</u> - some number of bytes before the start of the control block might be assigned to the block, so that you have fields at a negative displacement to the start of the control block

The CVT: The Ultimate Anchor

- One very special control block is the CVT (Communications Vector Table)
  - Many control block chains have their anchor in the CVT
  - The CVT also contains many system information fields of its own

The CVT: The Ultimate Anchor, 2

- The address of the CVT is pointed to from absolute address 16 (X'10') in memory
  - This has been true since the first release of OS/360 was available
- At the time your program is running, the first word of the CVT will have a pointer to the address of your job step task TCB (that is, a pointer to a pointer to the TCB)

The PSA: For Multiple CP Environments

- z Architecture mainframes come with multiple central processors (CPs), all sharing the same real storage
  - Since you can't allow any two to modify the same area of low storage at the same time, each CP has its own block of real storage allocated to virtual addresses 0-8192
    - Each CP has its own <u>prefix register</u> that does this mapping
- A data area called the Prefixed Save Area (PSA) maps the first 4K of memory for each CP

The PSA, fields

- Now the PSA has some interesting fields; to mention just a few:
  - Flags to indicate what hardware features are installed
  - Pointer to the current active TCB
  - Pointer to the current active ASCB
  - Pointer to the CVT

The PSA, General Use Programming Interface

- An interesting development is that the pointers to the current TCB and ASCB in the PSA are marked as GUPI
  - But the pointer to the pointer to the current TCB in the CVT is not indicated as GUPI
- This is interesting because all kinds of code, IBM, vendor, and user code, uses the CVT pointer to chase down control block information - so be careful, although this is unlikely to change

Some details of control blocks we've seen

- We have one true anchor / starting point:
  - 0 (x'00') is the beginning of PSA
  - PSA fields of interest (Offset, content):
    - 16 (x'10') - FLCCVT, pointer to CVT
    - 200 (x'C8') - FLCFACL, facilities installed flags
    - 540 (x'21C') - PSATOLD, address of current TCB
    - 548 (x'224') - PSAAOLD, address of current ASCB

More details of control blocks we've seen

- None of the fields in the CVT that are Programming Interfaces are of interest here

- None of the fields in the ASCB that are Programming Interfaces are of interest here

  - However, note that the field called ASCBRCTP at offset 124 (x'7C') does point to the RCT TCB for the address space - sometimes useful but not GUPI

Still more details of control blocks we've seen

- Fields in the TCB that are Programming Interfaces that are of interest here (Offset, content):
  - 0 (x'00') - TCBRBP, pointer to current RB
  - 116 (x'74') - TCBTCB, pointer to next TCB on queue of TCBs; zeros if last on the queue
  - 220 (x'DC') - TCBBACK, pointer to previous TCB on queue of TCBs; zeros if first on the queue
  - 132 (x'84') - TCBOTC, pointer to mother task's TCB (zeros for top task)
  - 136 (x'88') - TCBLTC, pointer to immediate daughter task's TCB (zeros for last task)

And more details of control blocks we've seen

- Fields in the PRB that are of interest here (Offset, content) [note - all RB fields are GUPI]:
    - 12 (x'0C') - RBCDE, pointer to the related CDE
        - Note that this contains one byte of flags and a 24-bit address, a common practice in the older control blocks
        - When chasing control blocks programmatically, if you are AMODE 31 (or AMODE 64), you need to ensure the flag byte is set to zeros before using it as an address

Even more details of control blocks we've seen

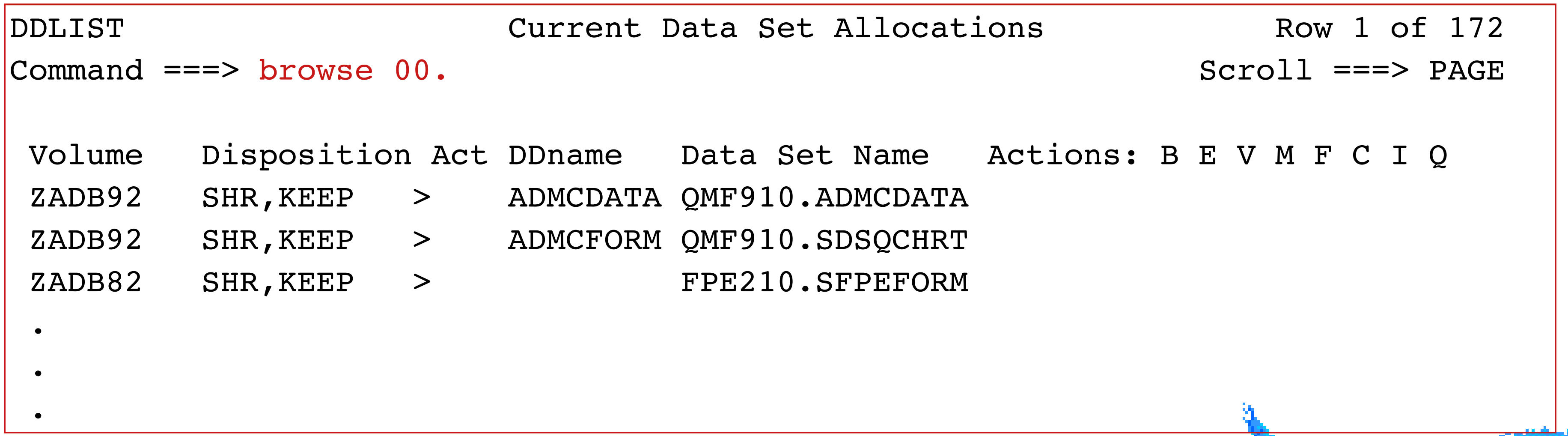
- Fields in the CDE that are of interest here (Offset, content) [note - all CDE fields are GUPI]:
  - 8 (x'08') - CDNAME, 8 byte name of the program represented by this CDE
    - If the result returned is "*PATHNAM" then the program resides in the HFS and to get the full name you need to call a system service such as CSVINFO or CSVQUERY

Using Control Blocks - Finding the data

- Once you know what control blocks you are interested in, now how to locate the blocks themselves? Here are some approaches:
  - ISPF / DDLIST / browse
  - SYSxDUMP DD statement
  - Programmatically chase the control blocks using anchors and pointers

Using Control Blocks - Finding the data, 2

► From any panel in ISPF you can issue the command DDLIST; from the resulting panel you can browse any (non-protected) location in memory using the Browse command with an address:



```
DDLIST                         Current Data Set Allocations                 Row 1 of 172
Command ===> browse 00.                                                     Scroll ===> PAGE

 Volume   Disposition Act DDname   Data Set Name   Actions: B E V M F C I Q
 ZADB92   SHR,KEEP    >   ADMCDATA QMF910.ADMCDATA
 ZADB92   SHR,KEEP    >   ADMCFORM QMF910.SDSQCHRT
 ZADB82   SHR,KEEP    >            FPE210.SFPEFORM
 .
 .
 .
```

► When you specify a number followed by a decimal point, it is assumed to be a hex address

Using Control Blocks - Finding the data, 3

► So browsing from location 0 shows the PSA and you see something like this:

```
 DDLIST    STORAGE  Start:00000000                                                                  00.
 Command ===>                                                                              Scroll ===> CSR
********************************************************** Top of Data ************************************************************
     +0 (00000000)  000A0000 000130E1 00000000 00000000  00FDDDB8 00000000 7FFFF000 7FFFF000  * .......÷.........Ùù½....".0.".0. *
    +20 (00000020)  7FFFF000 7FFFF000 7FFFF000 7FFFF000  00000000 00000000 7FFFF000 7FFFF000  * ".0.".0.".0.".0.........".0.".0. *
    +40 (00000040)  00000000 00000000 00000000 00FDDDB8  00000000 00000000 000A0000 000140E1  * .............Ùù½.............. ÷ *
```

► The format is similar to a SYSUDUMP

► You can now browse relative to this location using a relative address: b +10?
(add x'10' to the current address and then use this as a 31-bit pointer: go to that location) and you see:

```
 DDLIST    STORAGE  Start:00FDDDB8                                                                 +10?
 Command ===>                                                                              Scroll ===> CSR
********************************************************** Top of Data ************************************************************
     +0 (00FDDDB8)  00000218 00FEB85C 00FDDD34 00FDE3A0  00000000 00FF5700 00FF0FAE 00FE6752  * .....Ú½*.Ùù..ÙTµ......ï....Þ.ÚÅê *
    +20 (00FDDDD8)  00FE64DC 01998CF0 813596A0 00FE9CE8  00F521D0 00FE6E78 0109184F 00FDE3C8  * .ÚÀü.rð0a.oµ.ÚæY.5.}.Ú>Ì...|.ÙTH *
    +40 (00FDDDF8)  00F49000 00FF8040 00FEB882 00000000  0A0307FE 00FDDD3C 00FDDB88 00000000  * .4°...Ø .Ú½b.......Ú.Ùù..Ùûh.... *
```

Using Control Blocks - Finding the data, 4

- You can make this more powerful by setting up a names file (must be sequential; allocate to DDNAME ISRDDN) - then you can request browse by name!

  - For example, if *yourid*.TR.#CBNAMES contains:

```
PSA      00.
CVT      PSA+10?
ascb     psa+224?
tcb      psa+21c?
prb      tcb?
cde      prb+c%
```

  - Then from the command line issue: ==> tso alloc fi(isrddn) da(tr.#cbnames) shr reu
  - Then from the DDLIST panel you can issue commands like:

```
==> b cvt
==> b cde
```

and so on!

Using Control Blocks - Finding the data, 5

- Addresses are specified in DDLIST as one of these:
  - Absolute: 1-8 hex digits followed by a period (*e.g.*: 3f78.)
  - Relative: 1-8 hex digits preceded by a plus sign (*e.g.*: +16)
  - Symbolic: 1-8 alphanumeric characters, first of which is alpha (*e.g.*: PSA) [note: case-insensitive]
  - Indirect: an absolute, relative, or symbolic address followed by a % to indicate a 24-bit address or a ? to indicate a 31-bit address (*e.g.*: psa+10?)

Using Control Blocks - Finding the data, 6

- Now, a richer version of the #CBNAMES file:

```
PSA     0.              start of it all
CVT     PSA+10?         start of CVT
ascb    psa+224?        location of ascb
tcb     psa+21c?        current tcb from the PSA
tcbpsa  psa+21c?        same field, different name
tcbascb ascb+7c?        RCT TCB from the ascb
prb     tcb?            firt RB on the chain from my TCB
stc     tcbascb+88?     started task control TCB, from RCT
init    stc+88?         initiator TCB
jst     init+88?        jobstep task TCB
prbjst  jst?            PRB from jobstep task TCB
tcbcvt  cvt??           current TCB from CVT: note double ?'s
cde     prb+c%          cde from current prb; note 24-bit address
mname   cde+8           module name from cde
```

  - Note that each 'b' command lays its output on top of the previous one
  - Consider doing a PF3 (end) after you're done with one display, to avoid deep nesting
    - But don't back out of the first 'b 0.' screen, or you'll have to get back in all over again

Using Control Blocks - Finding the data, 7

- More information available from:
  - DDLIST Help (F1 from the ddlist display)
  - ISPF User's Guide (Appendix G)
  - TSO/E Programming Guide publication - syntax of address specifications found under the TEST command write up

Using Control Blocks - SYSUDUMPs

- If your program abends and you have supplied a SYSUDUMP or similar DD statement you will see a formatted dump which makes it easier to see the data (although you will not see all control blocks in all cases):

```
JOB SCOMSTOT          STEP RUNIT          TIME 181025   DATE 09183    ID = 000    CPUID = 0A0001161247   PAGE 00000001
COMPLETION CODE       SYSTEM = 0C1       REASON CODE = 00000001

  PSW AT ENTRY TO ABEND   078D0000  B6B00030  ILC  02  INTC  0001
PSW LOAD MODULE              ADDRESS = 36B00000  OFFSET = 00000030
NAME=#CBDUMP

 ASCB: 00FBAD00
          +0000  ASCB..... ASCB     FWDP..... 00FC9700  BWDP..... 00FC8400  LTCS..... 00000000  SVRB..... 007FD6C0
          +0014  SYNC..... 000F680B IOSP..... 00000000  R01C..... 0000      WQID..... 0000      SAWQ..... 00000000
. . .

 TCB: 007FF048
          +0000  RBP...... 007FDAC8  PIE...... 00000000  DEB...... 007C1038  TIO...... 007C6FE8  CMP...... 940C1000
. . .
          +0070  FSA...... 00006008  TCB...... 00000000  TME...... 00000000  JSTCB.... 007FF048  NTC...... 00000000
          +0084  OTC...... 007FF350  LTC...... 00000000  IQE...... 00000000  ECB...... 007D4EEC  TSFLG.... 20
. . .
PRB: 007CFC10
         -0020  XSB...... 7FFFDEE0  FLAGS2... 00        RTPSW1... 078D0000  B6B00030          RTPSW2... 00020001
         -000C            7F7BB000  FLAGS1... 00000000  WLIC..... 00020001
         +0000  RSV...... 00000000  00000000            SZSTAB... 00110082  CDE...... 007FF008
```

Using Control Blocks - control block chasing

- Here is an example of finding the module name of the current task in Assembler:

```
      L     2,540         point to TCB
      L     2,0(,2)       point to PRB
      L     2,12(,2)      point to CDE
      ICM   2,8,zero      clear leftmost bits
      MVC   o_name,8(2)   grab name
.
.
.
zero  DC    f'0'
```

- Assembler reminder: L is an RX instruction: L R1,D2(X2,B2)
  - So "L 2,540" is interpreted as: L   2,540(0,0)

- When 0 is specified as a base register or an index register, the contents of R0 are not used, but rather a value of 0
  - So "L 2,540" means "load the contents of absolute location 540, for 4 bytes, into R2"

Control Block Chasing

- You can also chase control blocks in other languages:
  - Gilbert Saint-Flour has some sample code in COBOL and PL/I on his site (visit http://gsf-soft.com/sitemap.shtml and scroll down to the Free Software heading)
  - An interesting Australian company, Longpela Expertise, has a nice page on control blocks with examples in several languages at:

    `http://www.longpelaexpertise.com.au/ezine/CtBlksBeginners2.html`

The PSA, another application

- Use the PSA to find if a feature is installed
  - If installed, use it directly
  - If not, may have to simulate with a subroutine or just forego the functionality, *e.g.*:

```
TM    203,X'02'    test for Extended Translation Facility 3
JZ    EXTR_SUB
CU24  R4,R8
```

The study of control blocks and data areas

- Control blocks are pervasive and instructive
  - Learning about them enhances your understanding of how things work
  - And sometimes you get glimpses of what's coming down the road (maybe)

    - For example, the PSA includes an 8 byte pointer to the "XCVT" - maybe that's a future 64-bit address for a version of the CVT!

The study of control blocks and data areas, 2

- A lot about control blocks are part of the folklore and culture of mainframe programmers and systems programmers
- It would be a good idea to find a mentor who can pass this folklore on
- Learn how they look things up, what they consider most important, how to reference and use control blocks
- Become a block head!

The Trainer's Friend
INCORPORATED

6790 East Cedar Avenue, Suite 201
Denver, Colorado 80224
USA

http://www.trainersfriend.com
303.393.8716
Sales: kitty@trainersfriend.com
Technical: steve@trainersfriend.com

Contact us for a copy of the Assembler program used to test lecture points.