HP Data Management
Training Course

SE 337 – IMAGE/3000
Internal Design and Optimization

Student Workbook

HEWLETT
PACKARD

INFORMATION NETWORKS DIVISION
19420 Homestead Road, Cupertino, CA 95014

Printed in U.S.A. 7/84

Copyright

NOTICE

The information in this document is subject to change without notice.

HEWLETT-PACKARD PROVIDES THIS MATERIAL "AS IS" AND MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS) IN CONNECTION WITH FURNISHING, PERFORMANCE OR USE OF THIS MATERIAL WHETHER BASED ON WARRANTY, CONTRACT, OR OTHER LEGAL THEORY.

Some states do not allow the exclusion of implied warranties or the limitation or exclusion of liability for incidental or consequential damages, so the above limitation and exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

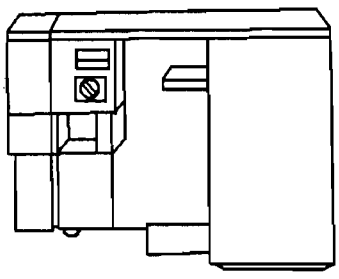
Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Preface

LASER PRINTED WORKBOOK



This workbook was printed on an HP 2680 Laser Printer. The graphics were prepared on an HP graphics terminal using HPDRAW and interfaced to the laser printer by TDP/3000, a Hewlett-Packard text processor.

Course Outline

Module 1

Overview

Module 2

Structures of IMAGE Components

Module 3

Establishing and Terminating Access Paths for the Data Base

Module 4

Reading Data Set Entries

Module 5

Modifying Data Set Entries

Module 6

Shared Access Considerations

Module 7

Transaction Logging and Intrinsic Level Recovery

Module 8

IMAGE with MPE V/P and MPE V/E

Appendix A

IMAGE Root File Layout

Appendix B

IMAGE Performance/Design Considerations

Course Objectives

The SE will be able to:

A. Demonstrate an understanding of the internals of IMAGE by successfully completing labs and quizzes throughout the course.

B. Identify and recommend appropriate data base and application design guidelines to optimize IMAGE performance by answering correctly 90% of the post test on customer problems.

Module 1.1

Overview of IMAGE Internals

Module 1.2

Module Outline

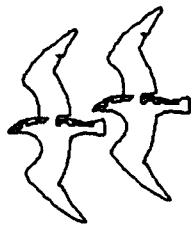
Overview

0. Introduction
1. Root File
2. Data Base Control Block (DBCB)
3. User Local Control Block (ULCB)
4. ILR Control Block (ILCB)
5. Remote Data Base Control Block (RDBCB)
6. Data Sets
7. Log Files

□ Overview

□ Notes

Overview of IMAGE Internals

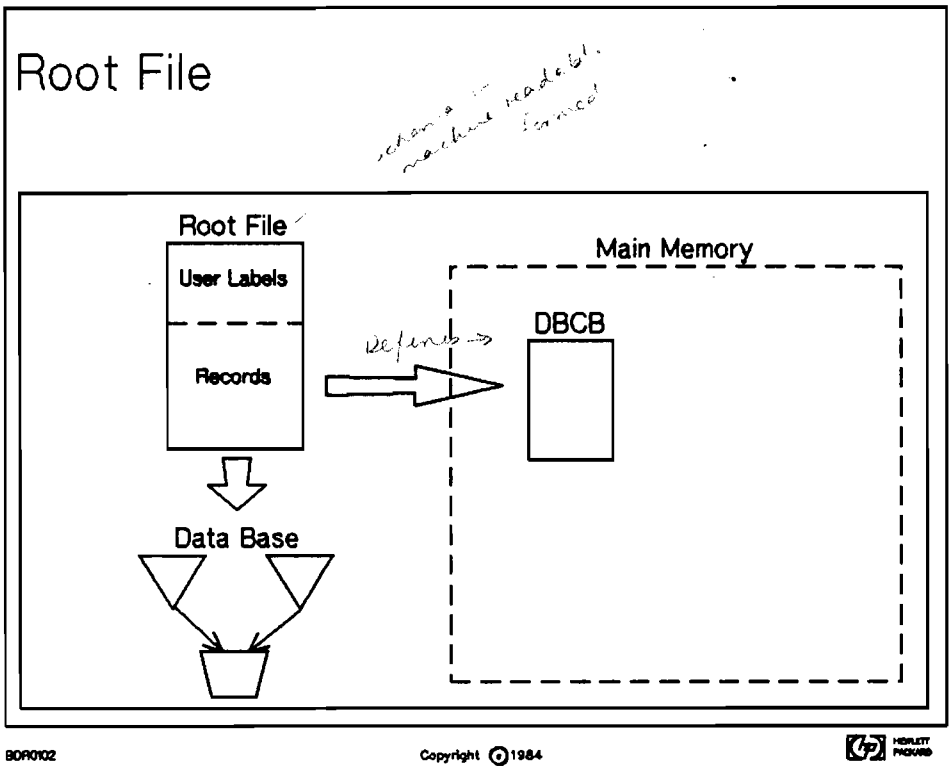


Root File DBCB ULCB ILCB RDBCB Data Sets Log Files

Module 1.4

☐ Overview

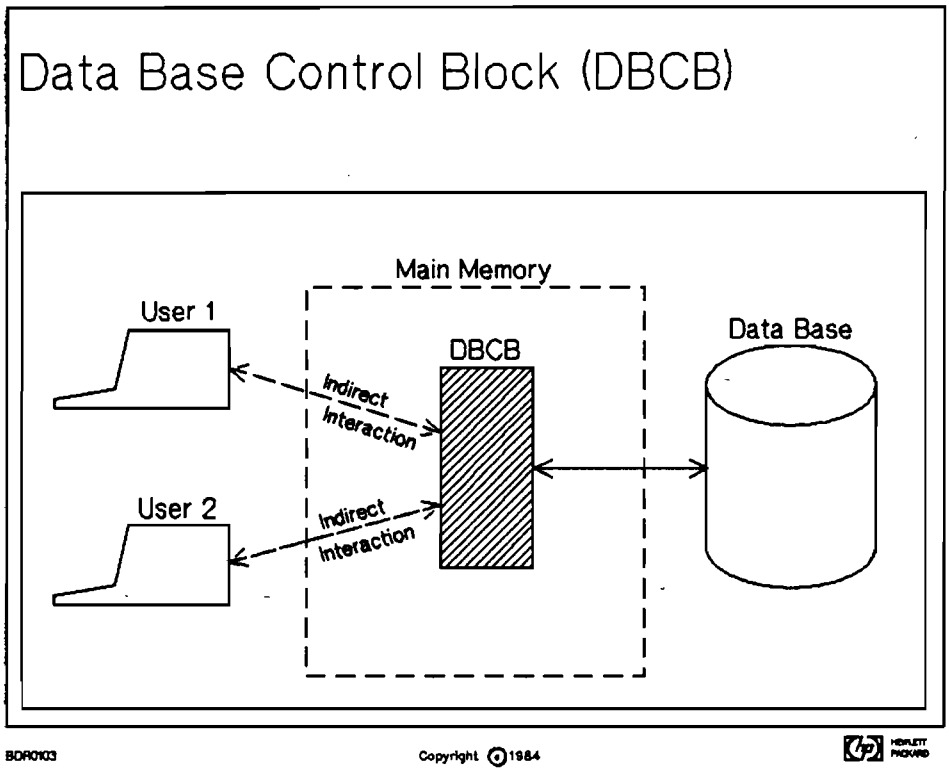
☐ Notes



- Contains data base schema in machine-readable form.
- Defines data base structure.
- Defines DBCB layout.
- User defined labels (user labels) used for security definitions.
- Records used for item, set definitions.

□ Overview

□ Notes

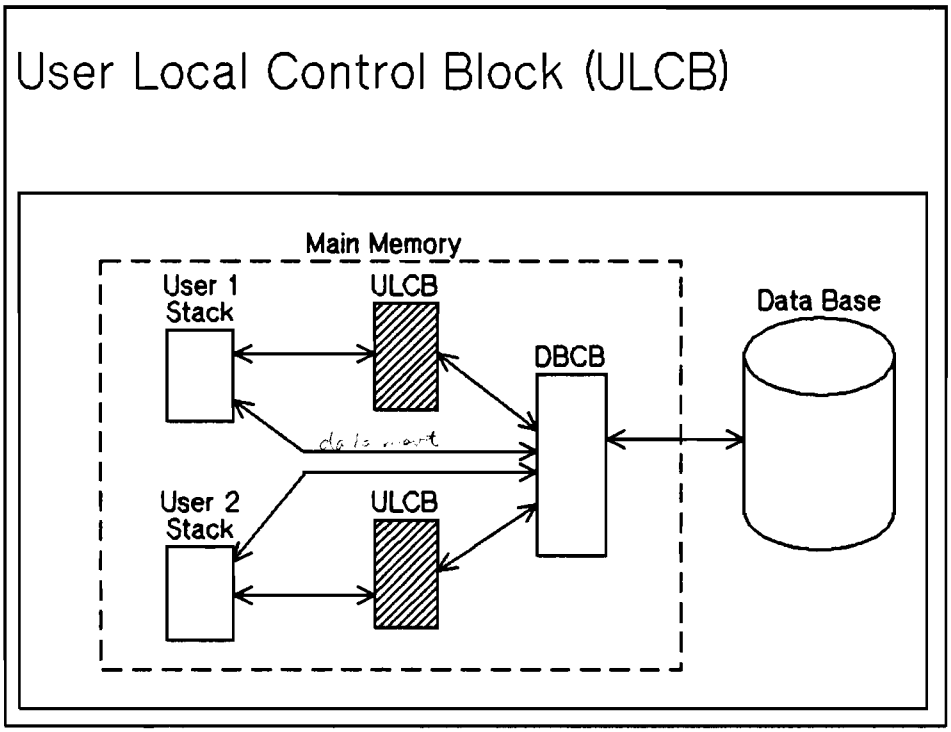


- Everyone must go through DBCB to access data base.
- One per open data base regardless of the number of concurrent users.
- Created by IMAGE when the first user opens a data base / released when the last user closes a data base.
- Contains global declarations, root file information, pointers, and buffers.

□ Overview

□ Notes

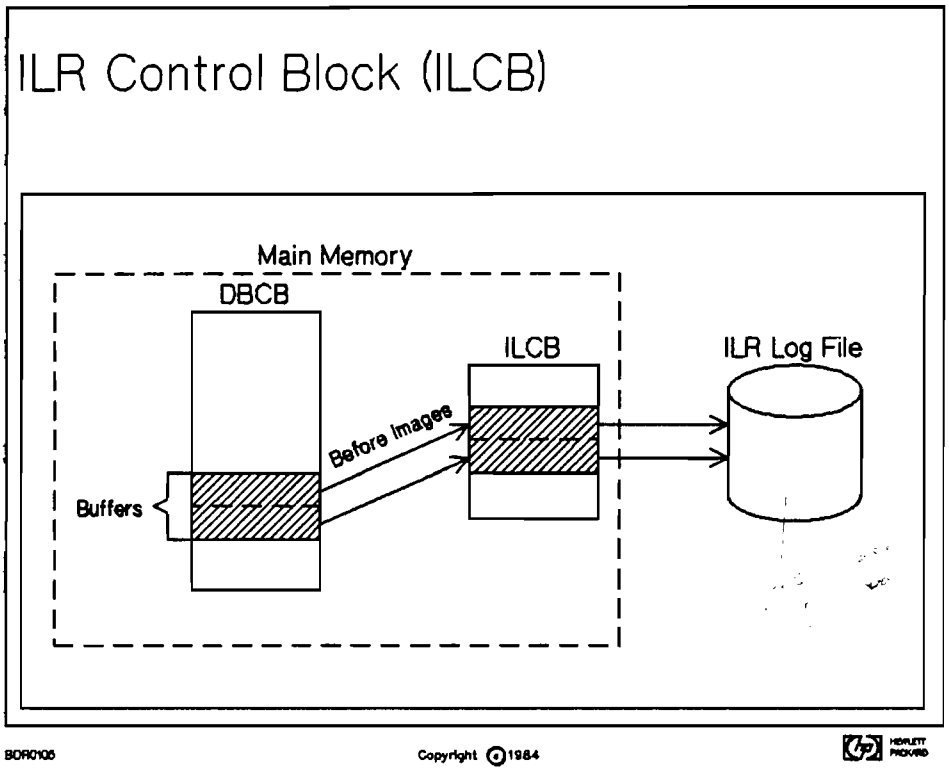
User Local Control Block (ULCB)



- A new ULCB is created when any process successfully calls DBOPEN / released when the process issues DBCLOSE.
- Establishes a new access path to the data base.
- Contains information local to each access path to the data base.

□ Overview

□ Notes

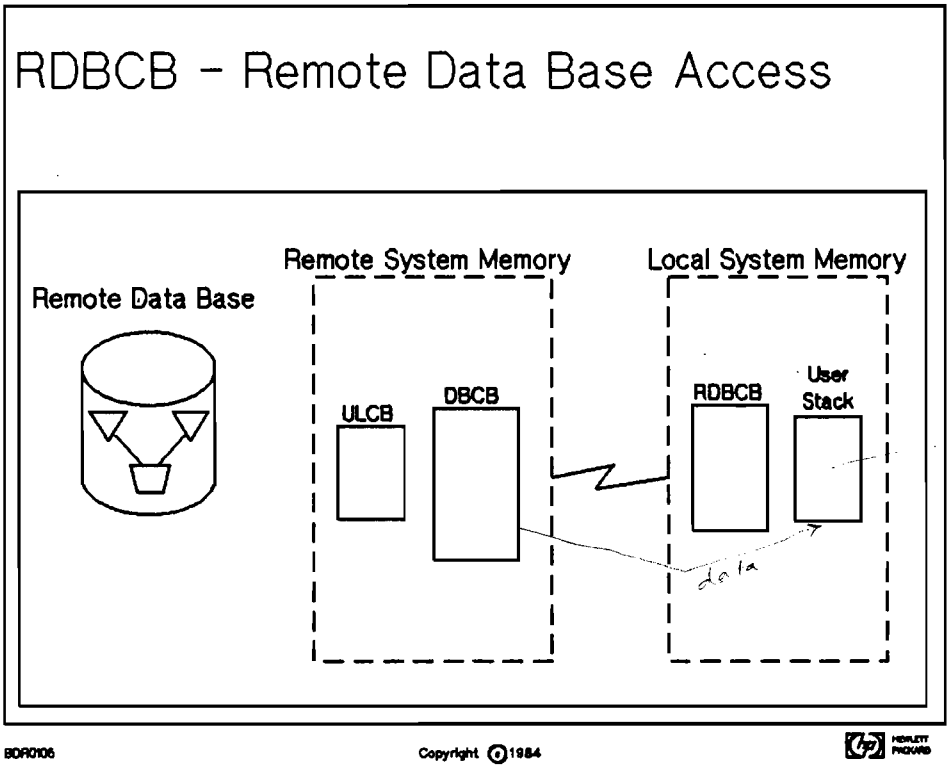


- Intermediate staging area for buffers to be modified by DBPUT and DBDELETE.
- Allocated upon first DBOPEN of the data base.

Module 1.8

□ Overview

□ Notes

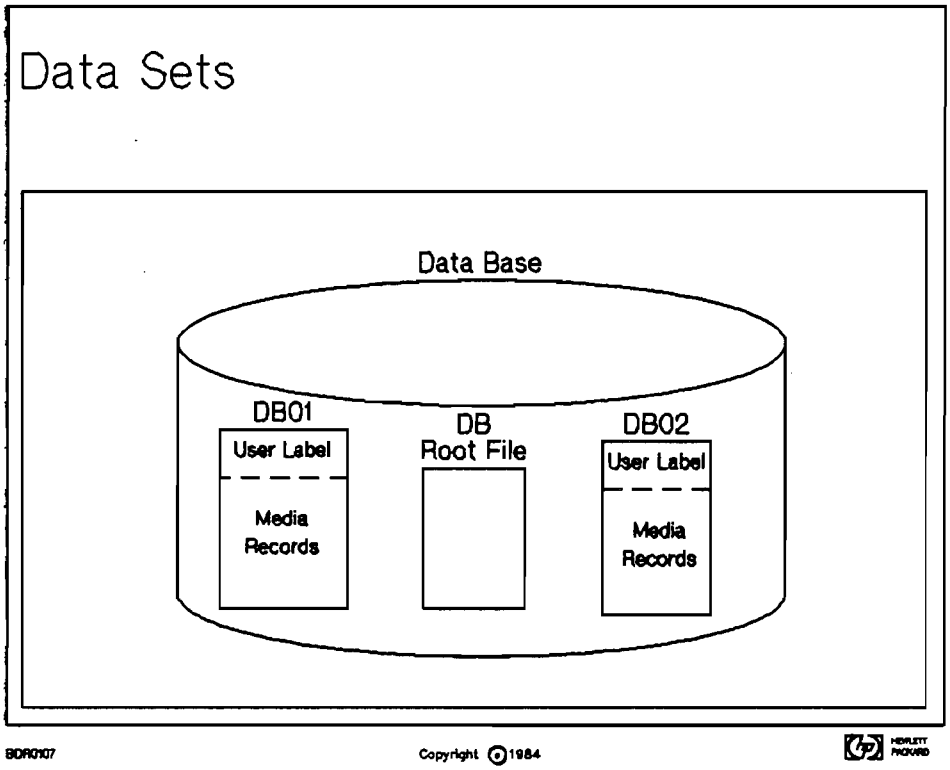


DBOPEN and calls — treated as

- 1 RDBCB per remote DBOPEN.
- ULCB and DBCB reside on remote system (where data base resides).
- Parsing, security checking done on local system - then call is passed to the remote system for processing.

□ Overview

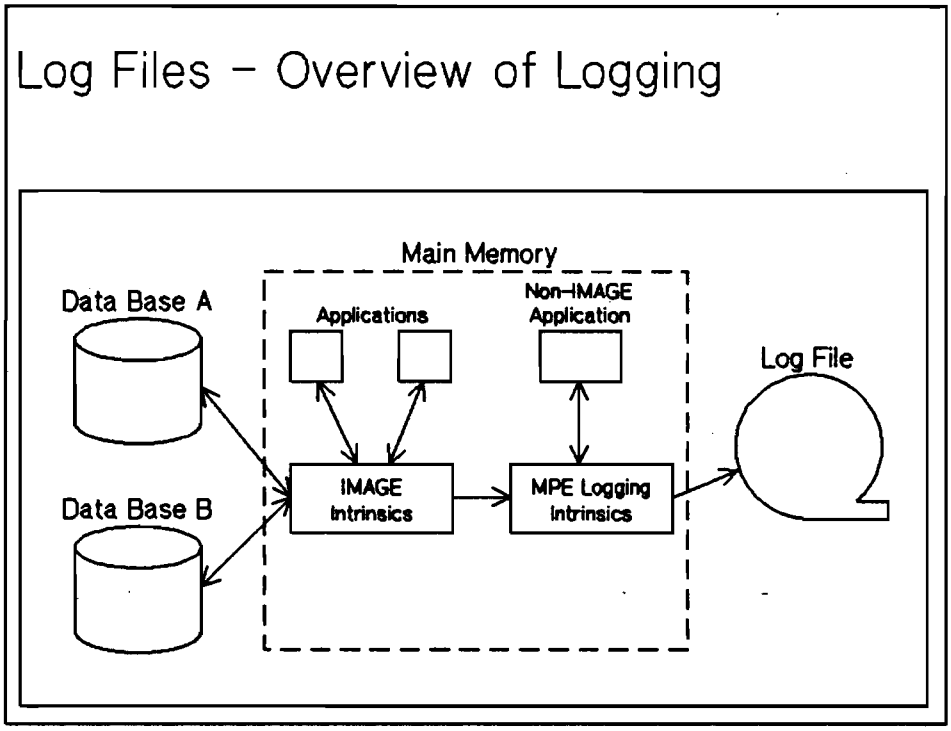
□ Notes



- MPE disc files containing media records.
- User defined labels (user labels) used for keeping track of free space and end of file.
- Media records contain data and pointer (chain) information.
- Priviledged files as indicated by negative file code.

□ Overview

□ Notes



- Transaction logging is a file system (MPE) feature.
- All data base modifications are logged by IMAGE using MPE Logging Intrinsics.
- May log to tape or disc.
- May log multiple data base (and can include other types of files) to a single log file.

Module 2.1

Structures of IMAGE Components

Module 2.2

Module Outline

Structures of IMAGE Components

0. Introduction
1. Root File Layout
2. Data Set Layouts

Worksession

3. Effects of Global DBCBs

Quiz

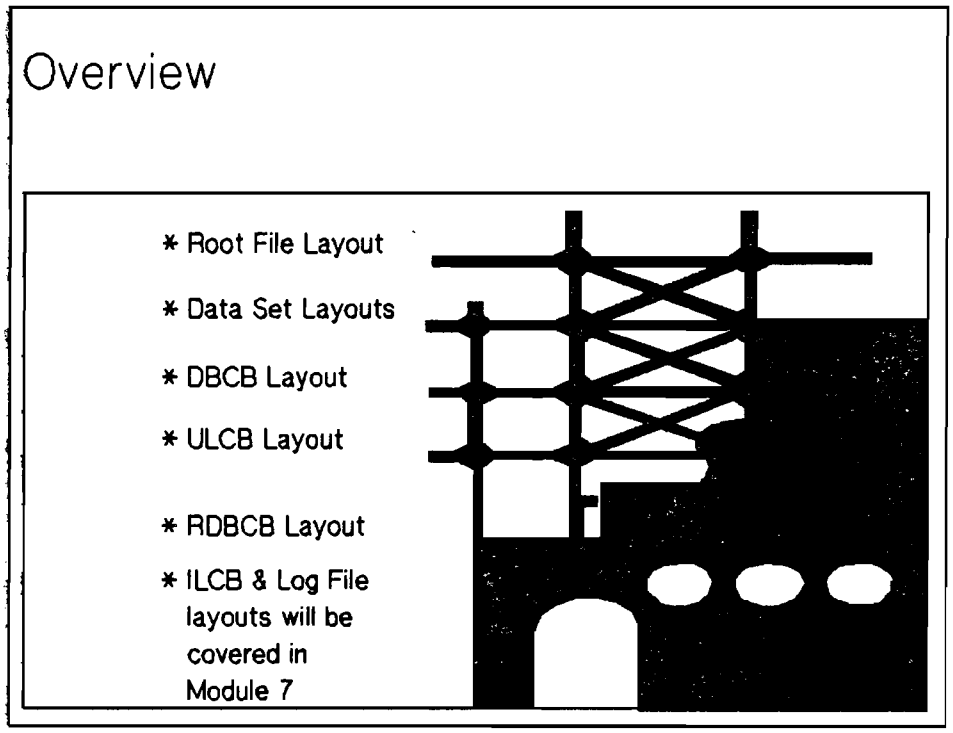
4. ULCB Layout
5. DBCB Layout
6. RDBCB Layout

Quiz

Module 2.3

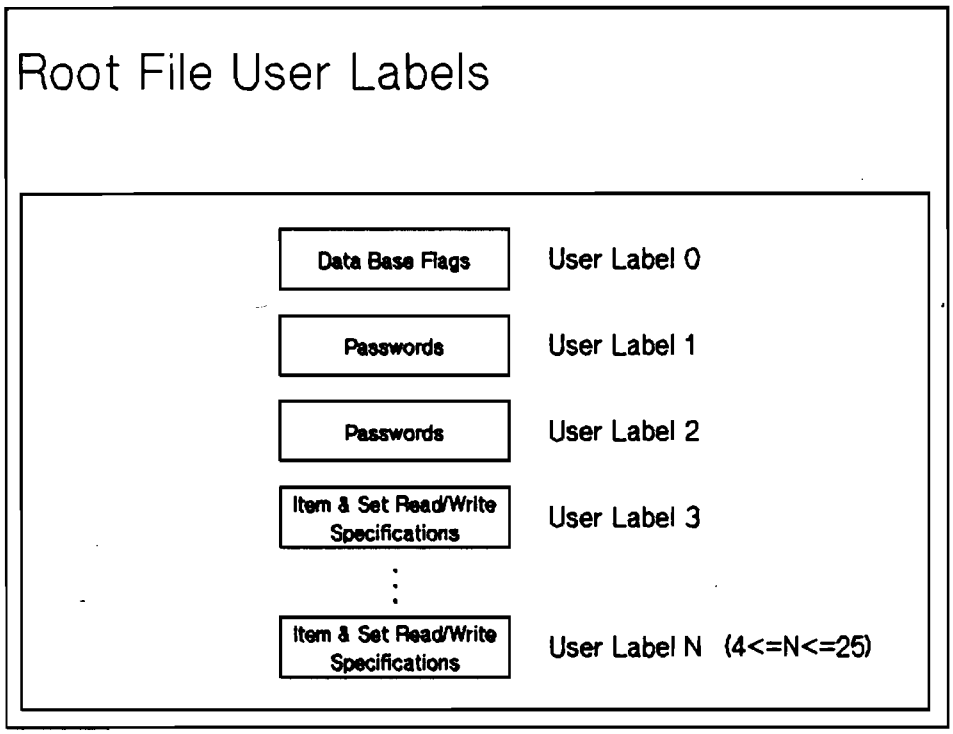
☐ Overview

☐ Notes



□ Root File Layout

□ Notes



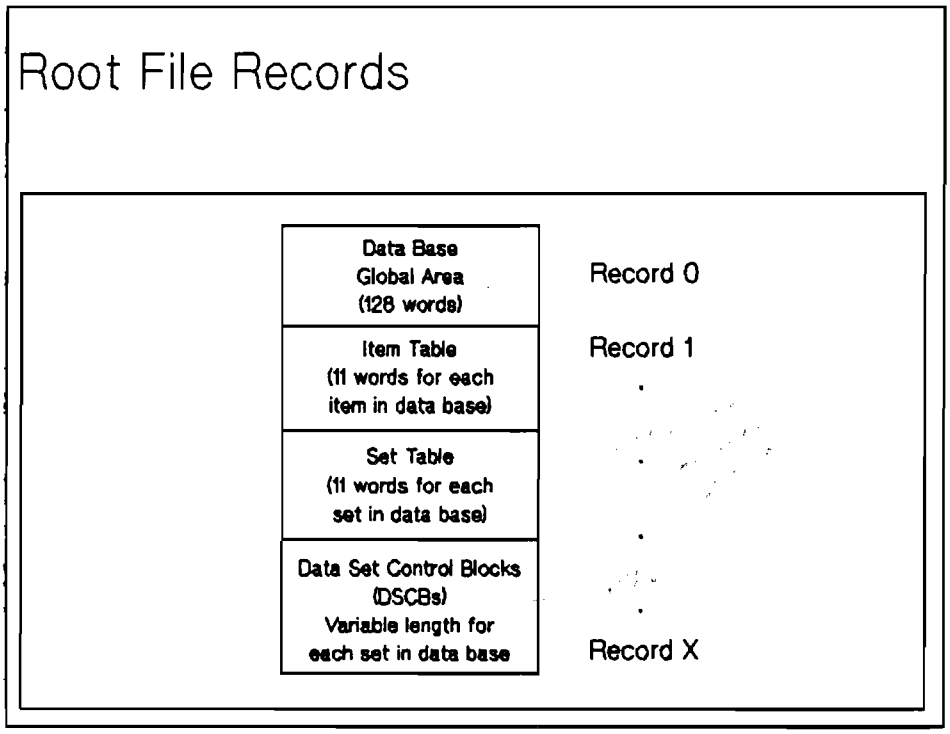
Each user label is 128 words long.

Passwords each take up 4 words.

Item and set read/write specifications each take up to 8 words.

☐ Root File Layout

☐ Notes



Item table contains information for each item such as the name, type, and length.

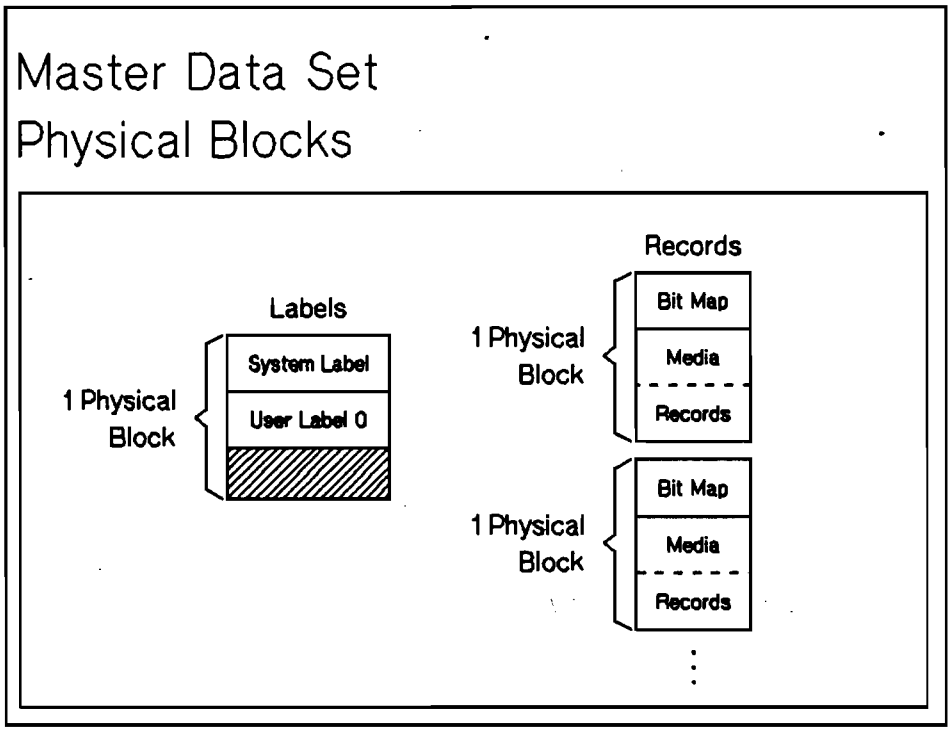
Set table contains information for each set such as the name, type, and DSCB pointer.

DSCBs contain information for each set. They are each broken down into:

1. Data set global area - capacity, lengths, counts.
2. Record definition tables - field count.
3. Path table - search and sort item(s), data set information.

□ Data Set Layouts

□ Notes



User Label 0

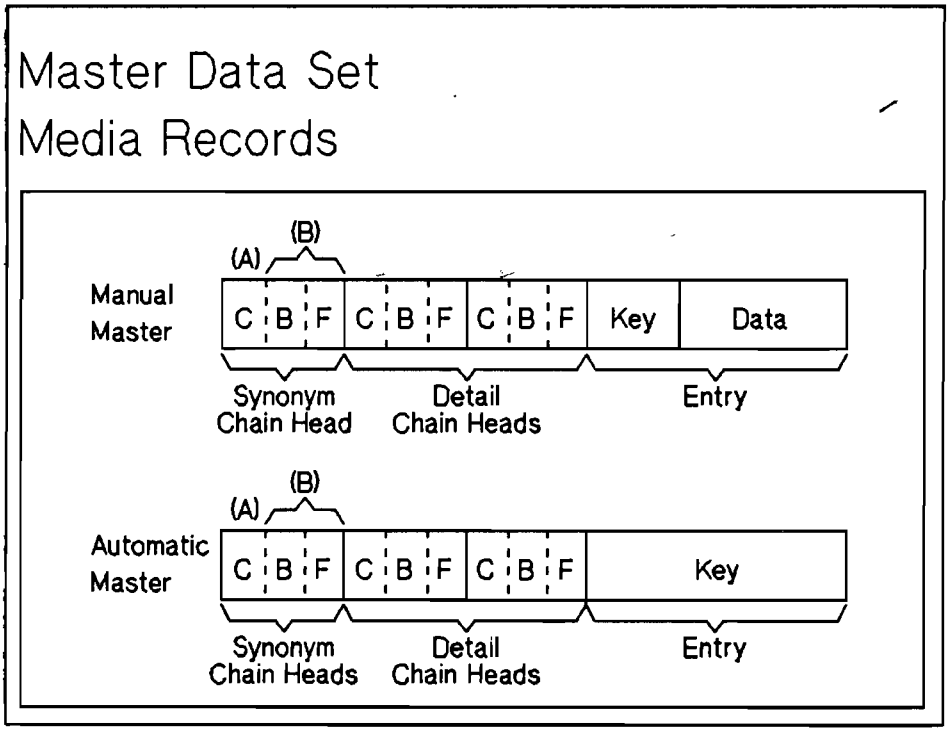
- Free space counter (2w).
  Incremented when an entry is deleted.
  Decremented when an entry is added.
- Data set capacity (2w).
- Data set EOF = file limit.

Data Set Blocks

- Bit map.
  At the beginning of each block.
  1 bit per record in the block.
  Used when adding a new entry.
    * 0 - primary, add record, set to 1.
    * 1 - secondary, cyclic search.
  Set to 0 when the corresponding entry is deleted.
- Media record.

□ Data Set Layouts

□ Notes



Synonym Count (A)

1. If = 1 & forward and backward synonym pointers (B) = 0
   - Indicates a primary entry.

2. If > 1 this indicates a primary entry.
   - Forward and backward synonym pointers indicate the address of the first and last synonym chain entries.

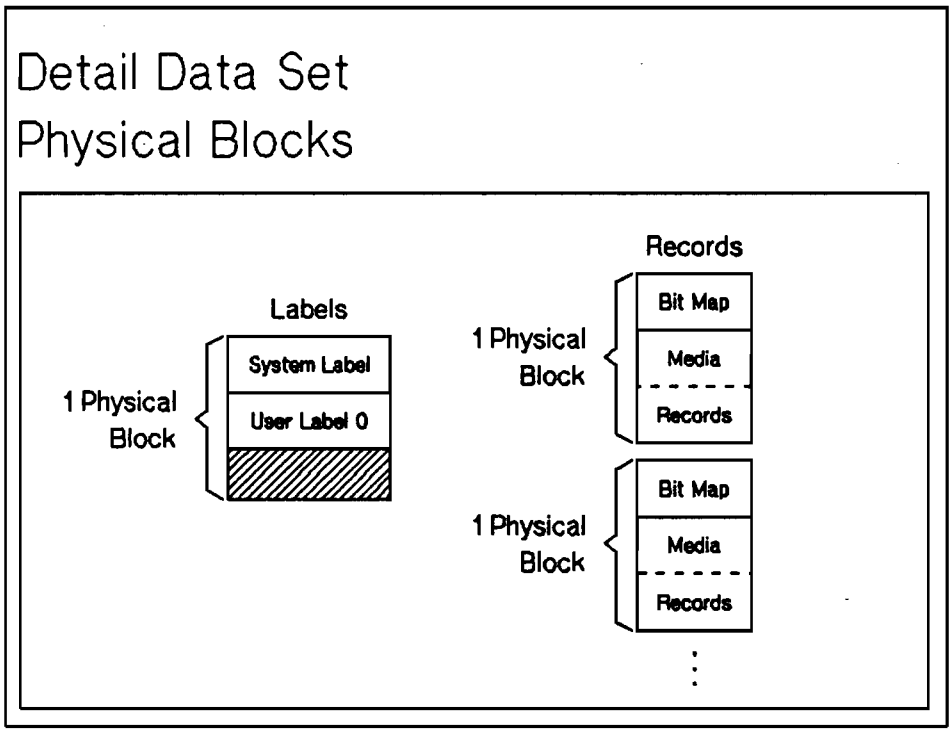
3. If = 0 & forward and backward synonym pointers ≠ 0.
   - Indicates a secondary.
   - With forward and backward synonym links.

Counts are one word each.

Pointers are two words each.

□ Data Set Layouts

□ Notes



User Label 0

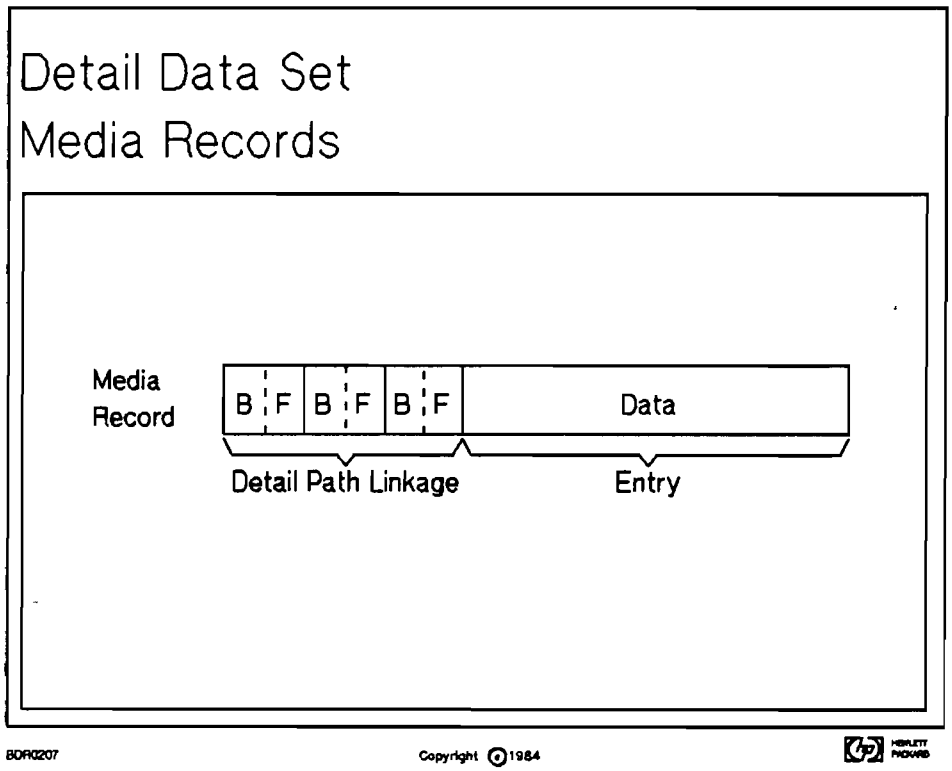
- Free space counter (2w).
- Data set EOF (2w).
- Free chain head (2w).
  Contains address of entry most recently deleted.
  IMAGE checks chain head when adding an entry.

Data Set Blocks

- Bit map.
  At the beginning of each block.
  1 bit per record in the block.
  Used for serial reading.
- Media records.

□ Data Set Layouts

□ Notes



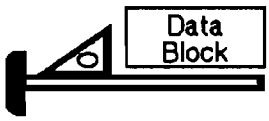
Forward and Backward Pointers

- Indicate the next & preceding entries on the detail chain.
- Two words each.

☐ Data Set Layouts

☐ Notes

Data Block Size (in words)



Block Size=(Media Record Size)*BF+(BF+15)/16
where BF is the blocking factor

Master Media Record Size=5+5*(number of paths)+(entry size)

Detail Media Record Size=4*(number of paths)+(entry size)

□ Data Set Layouts

□ Notes

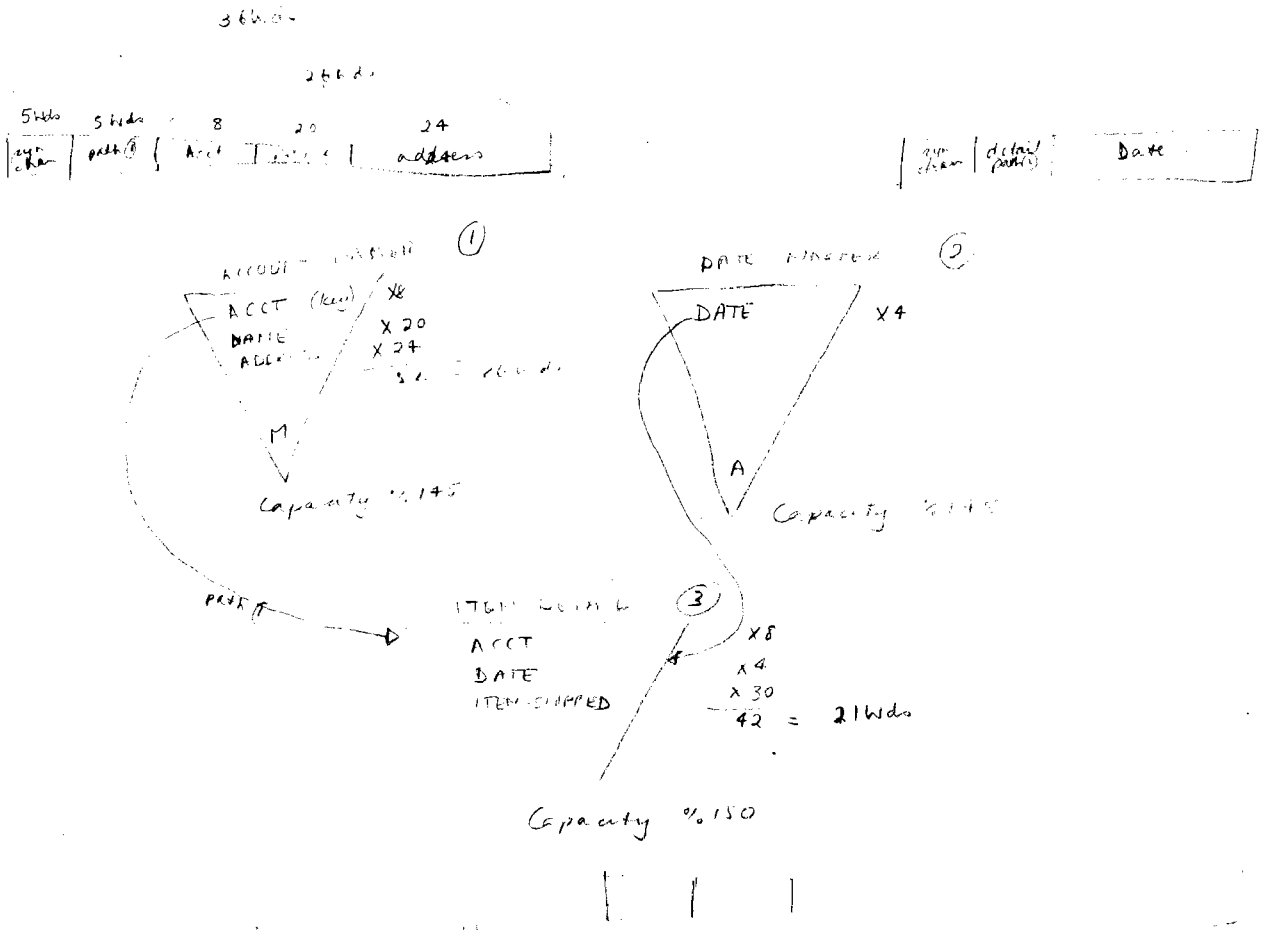
Dumping File Contents

* :RUN DBDUMP.PUB.SYS
* Program asks for:
  Filename (Data Set or Root File)
  Range (Starting Record : Number of Records)
* Starting record of -1 will dump all user labels
  (passwords, maintenance word, etc.)
* Dump is record-by-record in both octal & ascii
* Output defaults to line printer (formal designator "OUT")
  Use file equation to list on terminal
* Program requires Privileged Mode to use -1
* Enter /E to exit program
* Possible uses:
  checking broken chains, debugging, determining passwords/maintenance words, checking free space/synonym chain length in masters

Module 2.12

Worksession

1. Diagram the example data base using the root file dump given.
2. Draw the media record layout for each of the following media records.
   - A. Account - Master
   - B. Date - Master
   - C. Item - Detail



CC - CLERKS

SHPCLERK

□ Effects of Global DBCBs

□ Notes

Global DBCBs vs. Exclusive DBCBs

* Reduces total memory usage in multi-terminal applications
  - Root information which would be redundantly kept for each user is now kept once, in the DBCB
  - Buffers for all users kept in the DBCB
  - Specific user data kept in user's ULCB
  - Reduces load on Memory Manager
    Total memory requirements are decreased (reduces swapping)
    ULCBs are small, so easier to place in memory
    DBCBs may be "core resident" if all users accessing the same data base

Global DBCBs vs. Exclusive DBCBs

* Reduces explicit disc I/O performed by IMAGE
  - DBCB contains the only set of buffers for that data base
  - DBCB properly reflects the current contents of the data base

* Provides for automatic inter-process coordination fo buffer and locking
  - Utilizes record level locking mechanism
  - IMAGE automatically resolves (within individual intrinsic calls) the potential conflicts that used to be resolved by locking

* More buffers available to the individual user

Module 2.14

QUIZ

1. What are some of the functions of the data base global area? Version of IMAGE, various pointers

2. Data set control blocks are made up of the definitions of the data sets.

3. What information does user label 0 of the root file contain? Data base class; condition of DB, maintenance word, buffer specifications

4. What are the three areas of the DSCB?

   Data set global area

   Record definition table

   Path table

5. The user label of a master data set contains

   a Free Space Counter,

   a Data set Capacity,

   and a Data set EOF.

6. Why is the use of migrating secondaries essential? So that entries whose primary place is occupied by a secondary can occupy the correct record.

7a. How many chain heads will be present in a master entry if there are 12 paths from the master data set? 12 + 1 [illegible]

7b. How many words will make up this record (assume 20 words of data)? 13 x 5 + 20 = 85 wds

8. If there are 33 records per block, how many words will be necessary for the bit map? 3 Wds

9. What does IMAGE check first in Label 0 before adding a detail entry? Last deleted record number ie Free Chain Head.

QUIZ (cont.)

10a. What do the forward and backward pointers of a detail chain indicate?

10b. If both pointers are 0, what does this indicate?

11. What are the two control blocks used by IMAGE?

12. Which control block is created when the first person opens the data base? DBCB

13. What is the ULCB primarily used for?

14. What are some of the advantages of the DBCB?

15. If two users open a DB, how many DBCBs and ULCBs will be created?

16. Can a user accidentally modify another user's pointers? Explain.

ROOT FILE EXAMPL.IMAGE.KANE USER LABEL %000000

0	043127	124236	005026	016400	000000	021772	000000	000000	FW.....#.....
10	020040	020040	020040	020040	020040	020040	020040	020040	
20	020040	020040	020040	020040	000170	020040	020040	020040	.x
30	020040	020040	020040	020040	020040	020040	020040	020040	
40	020040	020040	020040	020040	020040	020040	020040	020040	
50	020040	020040	020040	020040	020040	020040	020040	020040	
60	020040	020040	020040	020040	020040	020040	020040	020040	
70	020040	020040	020040	020040	020040	020040	020040	020040	
100	035473	020040	020040	020040	004010	004411	005012	005413	;;A.....
110	006014	006415	007016	007417	010020	010421	010421	010421	.....
120	010421	010421	010421	010421	010421	010421	010421	010421	.....
130	010421	010421	010421	010421	010421	010421	010421	010421	.....
140	010421	010421	010421	010421	010421	010421	010421	010421	.....
150	010421	010421	010421	010421	010421	010421	010421	010421	.....
160	010421	010421	010421	010421	010421	010421	010421	010421	.....
170	010421	010421	010421	010421	010421	010421	010421	010421	.....

ROOT FILE EXAMPL.IMAGE.KANE USER LABEL %000001

0	020040	020040	020040	020040	020040	020040	020040	020040	
10	020040	020040	020040	020040	020040	020040	020040	020040	
20	020040	020040	020040	020040	020040	020040	020040	020040	
30	020040	020040	020040	020040	020040	020040	020040	020040	
40	020040	020040	020040	020040	020040	020040	020040	020040	
50	020040	020040	020040	020040	047505	026503	046105	051113	OE-CLERK
60	020040	020040	020040	020040	020040	020040	020040	020040	
70	020040	020040	020040	020040	020040	020040	020040	020040	
100	020040	020040	020040	020040	020040	020040	020040	020040	
110	020040	020040	020040	020040	020040	020040	020040	020040	
120	020040	020040	020040	020040	020040	020040	020040	020040	
130	051510	050103	046105	051113	020040	020040	020040	020040	SHPCLERK
140	020040	020040	020040	020040	020040	020040	020040	020040	
150	020040	020040	020040	020040	020040	020040	020040	020040	
160	020040	020040	020040	020040	020040	020040	020040	020040	
170	020040	020040	020040	020040	020040	020040	020040	020040	

ROOT FILE EXAMPL.IMAGE.KANE USER LABEL %000002

0	020040	020040	020040	020040	042102	026501	042115	044516	DB-ADMIN
10	020040	020040	020040	020040	020040	020040	020040	020040	
20	020040	020040	020040	020040	020040	020040	020040	020040	
30	020040	020040	020040	020040	020040	020040	020040	020040	
40	020040	020040	020040	020040	020040	020040	020040	020040	
50	020040	020040	020040	020040	020040	020040	020040	020040	
60	020040	020040	020040	020040	020040	020040	020040	020040	
70	020040	020040	020040	020040	020040	020040	020040	020040	
100	020040	020040	020040	020040	020040	020040	020040	020040	
110	020040	020040	020040	020040	020040	020040	020040	020040	
120	020040	020040	020040	020040	020040	020040	020040	020040	
130	020040	020040	020040	020040	020040	020040	020040	020040	
140	020040	020040	020040	020040	020040	020040	020040	020040	
150	020040	020040	020040	020040	020040	020040	020040	020040	
160	020040	020040	020040	020040	020040	020040	020040	020040	
170	020040	020040	020040	020040	020040	020040	020040	020040	

ROOT FILE EXAMPL.IMAGE.KANE USER LABEL %000003

ITEMS

```
   0  177777  177777  177777  177777  000000  000000  000000  000000  ................
  10  177777  177777  177777  177777  000000  000000  000000  000000  ................
  20  177777  177777  177777  177777  000000  000000  000000  000000  ................
  30  177777  177777  177777  177777  000000  000000  000000  000000  ................
  40  177777  177777  177777  177777  000000  000000  000000  000000  ................
  50  177777  177777  177777  177777  000000  000000  000000  000000  ................
  60  177777  177777  177777  177777  000000  000000  000000  000000  ................
  70  177777  177777  177777  177777  000000  000000  000000  000000  ................
 100  177777  177777  177777  177777  000000  000000  000000  000000  ................
 110  177777  177777  177777  177777  000000  000000  000000  000000  ................
 120  177777  177777  177777  177777  000000  000000  000000  000000  ................
 130  177777  177777  177777  177777  000000  000000  000000  000000  ................
 140  177777  177777  177777  177777  000000  000000  000000  000000  ................
 150  177777  177777  177777  177777  000000  000000  000000  000000  ................
 160  177777  177777  177777  177777  000000  000000  000000  000000  ................
 170  177777  177777  177777  177777  000000  000000  000000  000000  ................

ROOT FILE EXAMPL.IMAGE.KANE USER LABEL %000004

   0  000020  001000  040000  000000  000000  000000  040000  000000  ....@.......@...
  10  177777  177777  177777  177777  000000  000000  000000  000000  ................
  20  000020  001000  040000  000000  000000  000000  040000  000000  ....@.......@...
  30  177777  177777  177777  177777  000000  000000  000000  000000  ................
  40  177777  177777  177777  177777  000000  000000  000000  000000  ................
  50  177777  177777  177777  177777  000000  000000  000000  000000  ................
  60  177777  177777  177777  177777  000000  000000  000000  000000  ................
  70  177777  177777  177777  177777  000000  000000  000000  000000  ................
 100  177777  177777  177777  177777  000000  000000  000000  000000  ................
 110  177777  177777  177777  177777  000000  000000  000000  000000  ................
 120  177777  177777  177777  177777  000000  000000  000000  000000  ................
 130  177777  177777  177777  177777  000000  000000  000000  000000  ................
 140  177777  177777  177777  177777  000000  000000  000000  000000  ................
 150  177777  177777  177777  177777  000000  000000  000000  000000  ................
 160  177777  177777  177777  177777  000000  000000  000000  000000  ................
 170  177777  177777  177777  177777  000000  000000  000000  000000  ................

ROOT FILE EXAMPL.IMAGE.KANE RECORD %000000

   0  041001  042530  040515  050114  020040  000400  000773  000516  B EXAMPL.......N
  10  000005  000003  000165  000254  020040  020040  020040  020040  .....U..........
  20  000000  000037  000000  000000  000000  000000  000000  000000  ................
  30  000000  000000  000000  000000  000000  000000  000000  000000  ................
  40  000000  000000  000000  000000  000000  000000  000000  000000  ................
  50  000000  000000  000000  000000  000000  000000  000000  000000  ................
  60  000000  000000  000000  000000  000000  000000  000000  000000  ................
  70  000000  000000  000000  000000  000000  000000  000000  000000  ................
 100  000000  000000  000000  000000  000000  000000  000000  000000  ................
 110  000000  000000  000000  000000  000000  000000  000000  000000  ................
 120  000000  000000  000000  000000  000000  000000  000000  000000  ................
 130  000000  000000  000000  000000  000000  000000  000000  000000  ................
 140  000000  000000  000000  000000  000000  000000  000000  000000  ................
 150  000000  000000  000000  000000  000000  000000  000000  000000  ................
 160  000000  000000  000000  000000  000000  000000  000000  000000  ................
 170  000000  000000  000000  000000  000000  000000  000000  000000  ................

ROOT FILE EXAMPL.IMAGE.KANE RECORD %000001

   0  040503  041524  020040  020040  020040  020040  020040  020040  ACCT
  10  000000  000130  000410  042101  052105  020040  020040  020040  ..X..DATE
  20  020040  020040  020040  000000  000130  000404  040504  042122  ......X..ADDR
```

5 items / 3 sets

Item	Name	Size
1	ACCT	1 X 8
2	DATE	1 X 4
3	ADDRESS	1 X 24
4	ITEM-SHIPPED	1 X 30
5	NAME	1 X 20

30	042523	051440	020040	020040	020040	020040	000000	000130	ESS ..X
40	000430	044524	042515	026523	044111	050120	042504	020040	..ITEM-SHIPPED
50	020040	000000	000130	000436	047101	046505	020040	020040	...X.NAME
60	020040	020040	020040	020040	000000	000130	000424	040503	...X..AC
70	041517	052516	052055	046501	051524	042522	020040	000000	COUNT-MASTER .
100	000115	000330	042101	052105	026515	040523	052105	051040	.M..DATE-MASTER
110	020040	020040	000000	000101	000400	044524	042515	026504	...A..ITEM-D
120	042524	040511	046040	020040	020040	000000	000104	000444	ETAIL ...D.$
130	000000	000145	000771	000044	000032	007003	000601	000374	...e...$.....
140	000000	003403	000000	000145	000000	000000	000000	000000	.....e.....
150	000000	000000	000000	000000	000000	000000	000000	000000	.....
160	000000	000000	000000	000000	000000	000000	000405	001400	.....
170	000000	000000	000012	000016	000030	000044	000400	001401	.....$....

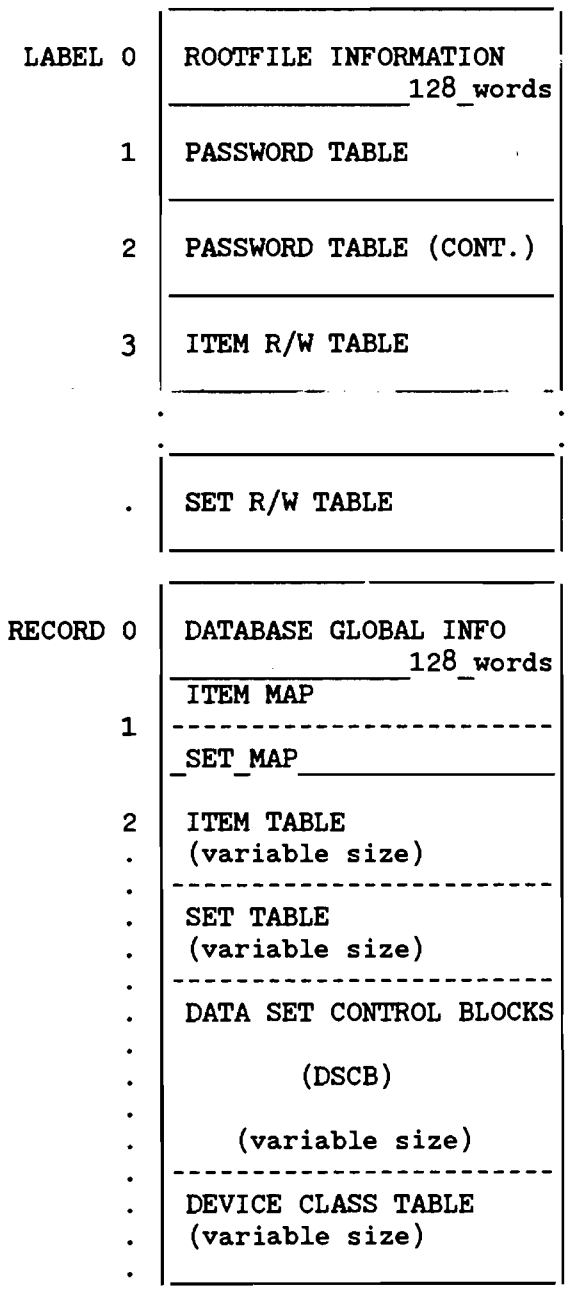
ROOT FILE EXAMPL.IMAGE.KANE RECORD %000002

0	000000	000145	000773	000014	000002	025001	000601	000440	...e.....*....
10	000000	001021	000000	000145	000000	000000	000000	000000	.....e.....
20	000000	000000	000000	000000	000000	000000	000000	000000	.....
30	000000	000000	000000	000000	000000	000000	001000	000000	.....
40	000000	000000	000014	001000	001402	000000	000150	000572	000035	.....h.z..
50	000025	006403	001001	000510	000000	003415	000000	000150	.....H.....h
60	000000	000000	000000	000000	000000	000000	000000	000000	.....
70	000000	000000	000000	000000	000000	000000	000000	000000	.....
100	000000	000000	000402	002000	000000	000000	000010	000014	.....
110	000016	000035	000400	000401	001000	001001	000000	000000	.....
120	000000	000000	000000	000000	000000	000000	000000	000000	.....
130	000000	000000	000000	000000	000000	000000	000000	000000	.....
140	000000	000000	000000	000000	000000	000000	000000	000000	.....
150	000000	000000	000000	000000	000000	000000	000000	000000	.....
160	000000	000000	000000	000000	000000	000000	000000	000000	.....
170	000000	000000	000000	000000	000000	000000	000000	000000	.....

ROOT FILE EXAMPL.IMAGE.KANE RECORD %000003

TURBO IMAGE

GENERAL ROOTFILE LAYOUT



The data base ROOT FILE is an MPE file with filecode equal to -400. The record size is 128 words, fixed, binary format with a blocking factor of 1. The size of the file depends on the number of data items and data sets defined in the data base.

Root File Label 0

			%
WORD 0	RL'CONDITION	(rootfile condition)	0
1	RL'DATE	(creation date)	1
2	RL'TIME	(creation time)	2
3			3
4	RL'EVEROPEN		4
5	RL'COLDLOADID	(cold load id)	5
6	RL'USERCOUNT		6
7	RL'DBG'IN'DBS	(DBG index in DBS)	7
8	RL'LOGID	(log id for transaction logging)	10
.			.
.			.
11			13
12	RL'LOGPASS	(log id password)	14
.			.
.			.
15			17
16	RL'FLAGS	(database flags)	20
17	RL'STORDATE	(DBSTORE date)	21
18	RL'STORTIME	(DBSTORE time)	22
19			23
20	RL'BUFSPECCOUNT	(buffer spec count)	24
21	RL'ILRCREATEDATE	(date ILR log created)	25
22	RL'ILRCREATETIME	(time ILR log created)	26
23			27
24	RL'ILRLASTDATE	(last log access date)	30
25	RL'ILRLASTTIME	(last log access time)	31
26			32
27	RL'RBPREDATE	(previous rollback date)	33
28	RL'RBPRETIME	(previous rollback time)	34
29			35
30	RL'RBDATE	(rollback date)	36
31	RL'RBTIME	(rollback time)	37
32			40
33	RL'LANGUAGE'ID	(language id)	41
34	RL'LANG'MNEMONIC	(language mnemonic)	42
.			.
.			.
41			51
42	RESERVED		52
. .	FOR		. .
. .	FUTURE		. .
63		USE	77
64	RL'MAINTWORD	(database maintenance	100
65		word)	101
66			102
67			103
68	RL'BUFFERSPECS	(buffer specifications)	104
			.
to			.
			.
127			177

Root File Label 0 (CONTINUED)

RL'CONDITION (IN ASCII):

```
        JB - Virgin. The database has not been created yet.
        FW - OK. The database is OK.
        RM - Modified deferred. The database is being modified.
        MC - Maintenance create. The database is being created.
        ME - Maintenance erase. The database is being erased.
        IL - ILR recovery in progress.
        IE - ILR enable in progress.
        ID - ILR disable in progress.
        CN - Conversion by DBCONV is in progress.
        MV - data set file moving is in progress.
```

RL'DATE: Root file creation date*. Its format is:

```
          0:_1:_2:_3:_4:_5:_6:_7: 8:_9:10:11:1    4:15
        |year_____________|day_of_year_____________|
```

RL'TIME: Root file creation time*. Its format is:

```
          0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15
        |hour_____________|minutes_____________|
        |seconds__________|tenth_of_seconds____|
```

RL'EVEROPEN: This field is no longer used under IMAGE B

RL'FLAGS:

```
        (0:1) - RECOVERY          Default is NO  (0)
        (1:1) - LOGGING           Default is NO  (0)
        (2:1) - ACCESS            Default is YES (1)
        (3:1) - DUMPING           Default is NO  (0)
        (4:1) - OUTPUT DEFER      Default is NO  (0)
        (5:2) - SUBSYSTEM ACCESS  Default is R/W (00)
        (7:1) - ILR               Default is NO  (0)
        (8:1) - ROLLBACK          Default is NO  (0)
        (9:1) - RESERVED-FOR-FUTURE-USE
        (10:1)- DIRTY FLAG        Default is YES (1).
                                  This indicates the database has
                                  been modified but not DBSTOREd.
        (11:1)- DBRECOV RESTART   Default is NO  (0)
        (12:1)- SWQ               Default is NO  (0)
        (13:3)- RESERVED-FOR-FUTURE-USE
```

RL'STORDATE: Same format as RL'DATE*.

RL'STORTIME: Same format as RL'TIME*.

RL'BUFSPECCOUNT: Maximum number of buffer specifications allowed.

Root File Label 0 (CONTINUED)

RL'ILRCREATEDATE: Same format as RL'DATE*.

RL'ILRCREATETIME: Same format as RL'TIME*.

RL'ILRLASTDATE: Same format as RL'DATE*.

RL'ILRLASTTIME: Same format as RL'TIME*.

RL'RBPREDATE: Same format as RL'DATE*.

RL'RBPRETIME: Same format as RL'TIME*.

RL'RBDATE: Same format as RL'DATE*.

RL'RBTIME: Same format as RL'TIME*.

RL'LANGUAGE'ID: Sama format as defined in systemconfiguration.

RL'LANG'MNEMONIC: Language mnemonic for this database.
Maximum 16 characters.

RL'MAINTWORD: For data bases with no maintenance word this field has
2 semicolons (';;') and trailing blanks.

RL'BUFFSPECS:

BIT/	0: 1: 2: 3: 4: 5: 6: 7: 8: 9:10:11:12:13:14:15		%
WD 68	buffers_for_1 user_____	buffers_for_2 users____	104
69	buffers_for_3 users____	buffers_for_4 users____	105
.	etc...		.
.			.
127	buffers_for_119 users__	buffers_for_120 users__	177

* The DATE and TIME fields can be formatted (for display purposes) individually by calling the FMTCALENDAR and FMTCLOCK Intrinsics respectivelly. Or both fields can be formatted at once with FMTDATE Intrinsic.

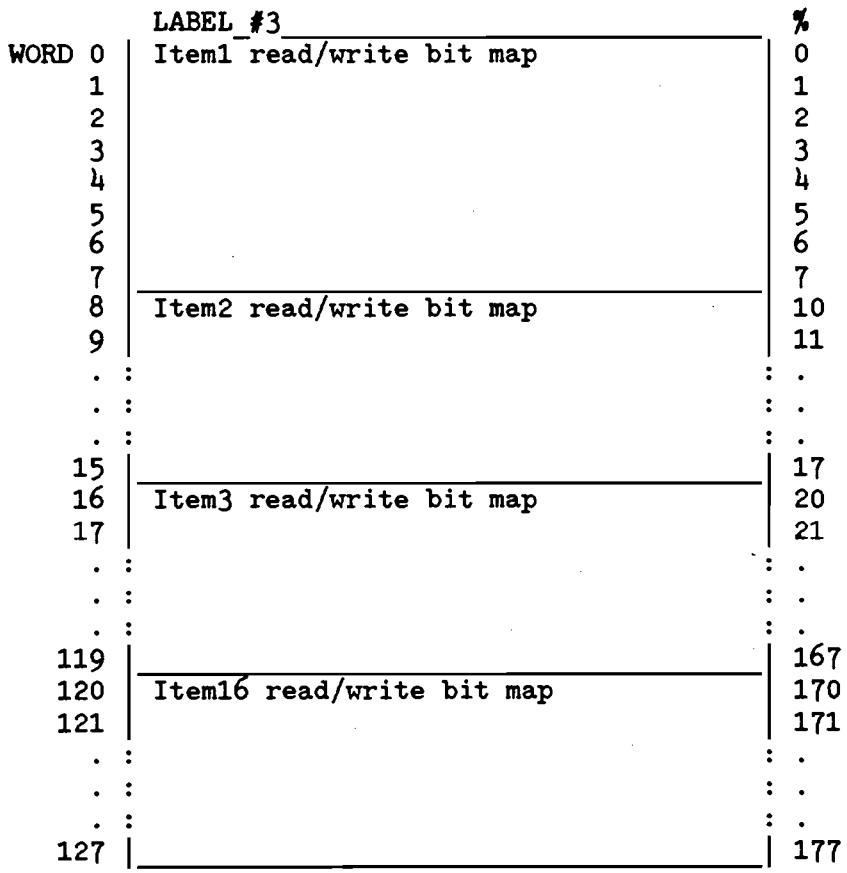
Root File Labels 1 & 2

	LABEL_#1	%
WORD 0	Password for user class 0	0
1	(this is a dummy field since user	1
2	class 0 is not defined)	2
3		3
4	Password for user class 1	4
5		5
6		6
7		7
8	Password for user class 2	10
9		11
10		12
11		13
.		.
.	.	.
.		.
124	Password for user class 31	174
125		175
126		176
127		177

	LABEL_#2	
0	Password for user class 32	0
1		1
2		2
3		3
4	Password for user class 33	4
5		5
6		6
7		7
8	Password for user class 34	10
9		11
10		12
11		13
.		.
.	.	.
.		.
124	Password for user class 63	174
125		175
126		176
127		177

The PASSWORD TABLE occupies user labels number 1 and 2.
There are four words (8 characters) reserved for each password. The relative position of a password corresponds to the user class number defined in the schema. For user class numbers not defined in the SCHEMA, the four word field is filled with blanks.

Root File Label 3



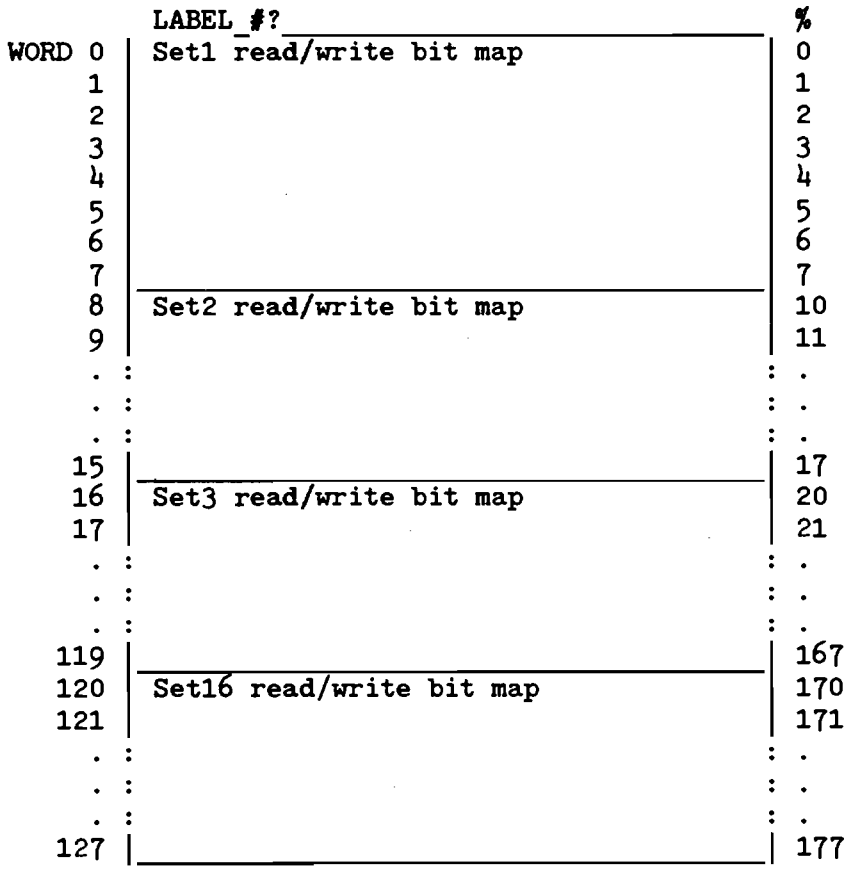
The ITEM READ/WRITE TABLE starts in user label #3
There are eight words for each ITEM READ/WRITE bit map.
For databases with more than 16 items, the read/write table continues in the next user labels. The specific format of this table is explained after the SET READ/WRITE TABLE since it is defined the same way.
The number of user labels occupied by the ITEM READ/WRITE TABLE depends on the number of data items defined in the schema and can be obtained by rounding upwards (ceiling)the result of:

Num-of-labels = [(Num-of-items)*8]/128

Since there can only be a maximum of 1023 data items in the schema, the maximum size for this table in user labels would be:

Max-size = [(1023)*8]/128 = 63.93 => 64 labels.

Root File- Next Label



The SET READ/WRITE TABLE starts on a user label boundary after the ITEM READ/WRITE TABLE.
There are eight words for each SET READ/WRITE bit map.
For databases with more than 16 data sets, the read/write table continues in the next user labels.
The specific format of this table is shown in the next page.

The number of user labels occupied by the SET READ/WRITE TABLE depends in the number of data sets defined in the schema, and is obtained by rounding upwards (ceiling) the result of:

Num-of-labels = [(Num-of-sets)*8]/128

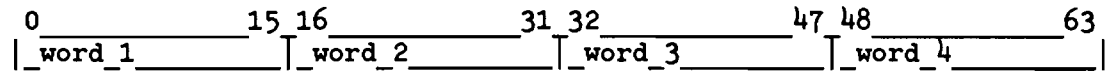
Since there can only be a maximum of 199 data sets defined in the schema the maximum size for this table in user labels is:

Max-size = [(199)*8]/128 = 12.44 => 13 labels

ITEM/SET READ/WRITE TABLE FORMAT

There are eight words per item/set read/write table definition and up to 16 items/sets per record (user label). Within each 8 words, the first 4 words are the flags for the user classes wich have read access to the item/set. The second 4 words are the flags for the user classes which have write access to the item/set. The detail format for an eight word field is shown below.

A. Four words for read access:

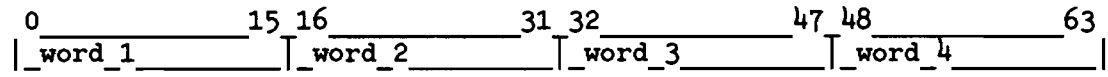


4 words represent 64 bits. Bit n represents read access for user class n to the item/set. If bit n is set to 1 then user class n has read access to the item/set.
For example, if the word settings are:

word 1	word 2	word 3	word 4
%000016	%020000	%000410	%001300

This means that user classes 12, 13, 14, 18, 39, 44, 54, 56 and 57 have read access to the item/set.
If no read/write security is defined at all for the item/set, then all of the read security bits are set to 1.

B. Four words for write access:



Write access flags have the same format as the read access flags. Bit n represents write access for user class n to the item/set. If bit n is set to 1, then user class n has write access to the item/set
For example, if the word settings are:

word 1	word 2	word 3	word 4
%000010	%020000	%000000	%001100

This means that the user classes 12, 18, 54 and 57 have write access to the item/set.
If no read/write security is defined at all for the item/set, then all of the write security bits are set to 0.

Root File Record 0

	RECORD #0		%
word 0	ROOT'DBSTATUS		0
1	ROOT'DBNAME		1
2			2
3			3
4			4
5	ROOT'TRLRLGTH	(trailer_area_length)	5
6	ROOT'BUFFLGTH	(buffer_length)	6
7	ROOT'LGTH	(rootfile length)	7
8			10
9	ROOT'ITEMCT	(number_of_items)	11
10	ROOT'SETCT	(number_of_data_sets)	12
11	ROOT'ITEMPTR	(record_#_of_item_table)	13
12	ROOT'DSETPTR	(record_#_of_set_table)	14
13	ROOT'DSCBPTR	(record_#_of_DSCBs)	15
14	ROOT'DEVICEPTR	(record_#_of_device_class_tbl)	16
15	ROOT'DBGFLAG		17
16	RESERVED	(set to blanks)	20
17			21
18			22
19			23
20	NOWOPEN		24
21	MAXOPEN		25
22	RESERVED	(for future use)	26
.	:	(set to binary 0s) :	.
.	:	:	.
.	:	:	.
127			177

ROOT'DBSTATUS - (0:8) IMAGE version ('C' in ASCII)
(8:8) Binary 2 (filler)

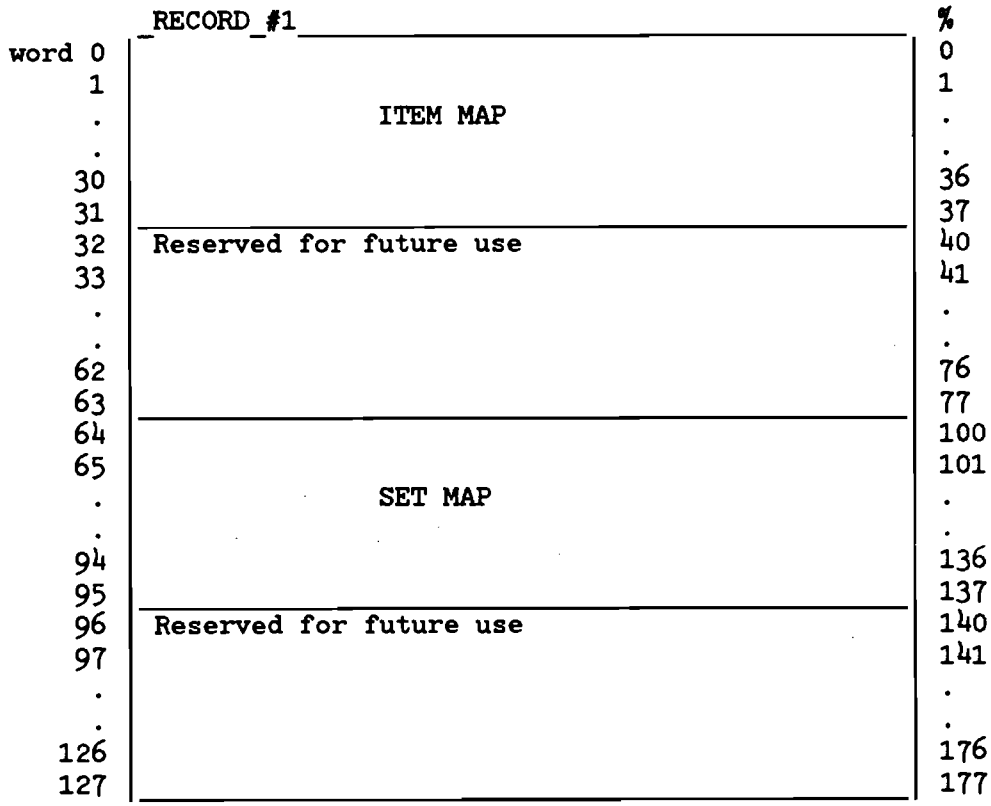
ROOT'DBNAME - DATABASE name left justified (last 2 chars are blank).

NOWOPEN - Number of data sets opened. This field is not used in IMAGE B & C

MAXOPEN - Maximum number of data sets that can be opened. This field is not used in IMAGE B & C.

ROOT'DBGFLAG - 1: Information can fit in DBG.
0: Information can not fit in DBG.

Root File Record 1



The ITEM MAP occupies words 0-31.

The SET MAP occupies words 64-95.

These two maps are used by DBOPEN for faster access to information in the ITEM TABLE and SET TABLE.

Root File Record 2

bits/ word	0:1:2:3:4:5:6:7:8:9:10:11:12:13:14:15	%
0	item-name-1	0
1		1
2		2
3		3
4		4
5		5
6		6
7		7
8	item-no-of-synonym	10
9	reserved-1 | reserved-2	11
10	item-type | subitem-count	12
11	subitem-length | (not used)	13
12	item-name-2	14
13		15
14		16
15		17
16		20
17		21
18		22
19		23
20	item-no-of-synonym	24
21	reserved-1 | reserved-2	25
22	item-type | subitem-count	26
23	subitem-length | (not used)	27
24		30
.	:	:

The ITEM TABLE starts in record #2.
Each entry is 12 words long and the length of the table depends on the number of data items defined in the schema. The realtive position of an item definition depends on its relative position in the schema.

Item-name: is a data item name, left-justified and with trailing blanks

Item-number-of-synonym: is the number of the item whose name has the same hashed result as this one (this is utilized for quick item name searches).

Item-type: is one of the following: I, J, K, R, X, U, Z, or P

```
                  item-type
                     |
        VALUES, 20J2;
                  |  |subitem-length
                  |subitem-count
```

The maximum size for this table is 12*1023 = 12276 words

NOTES:
The reserved-1 and reserved-2 fields are the 'old' level numbers for read and write security. Now, the values are always zero.

Root File- Next Record

```
bits/   _0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15  %
word 0 | set-name-1                                      | 0
     1 |                                                | 1
     2 |                                                | 2
     3 |                                                | 3
     4 |                                                | 4
     5 |                                                | 5
     6 |                                                | 6
     7 |________________________________________________| 7
     8 | set-no-of-synonym_____ | reserved-1___________ | 10
     9 | reserved-2____________ | data-set-type________ | 11
    10 | DSCB-pointer                                   | 12
    11 |________________________________________________| 13
    12 | set-name-2                                     | 14
    13 |                                                | 15
    14 |                                                | 16
    15 |                                                | 17
    16 |                                                | 20
    17 |                                                | 21
    18 |                                                | 22
    19 |________________________________________________| 23
    20 | set-no-of-synonym_____ | reserved-1___________ | 24
    21 | reserved-2____________ | data-set-type________ | 25
    22 | DSCB-pointer                                   | 26
    23 |________________________________________________| 27
    24 |                                                | 30
     . :                                                :
```

Set table follows the Item Table.

Each entry is 12 words long. The length of the table depends on the number of data sets defined in the schema. The relative position of a set definition depends on its relative position in the schema.

Set-name: is a data set name, left-justified and with trailing blanks.

Set-number-of-synonym: is the number of a data set whose name has the same hashed result as this one (this is utilized for quick set name searches).

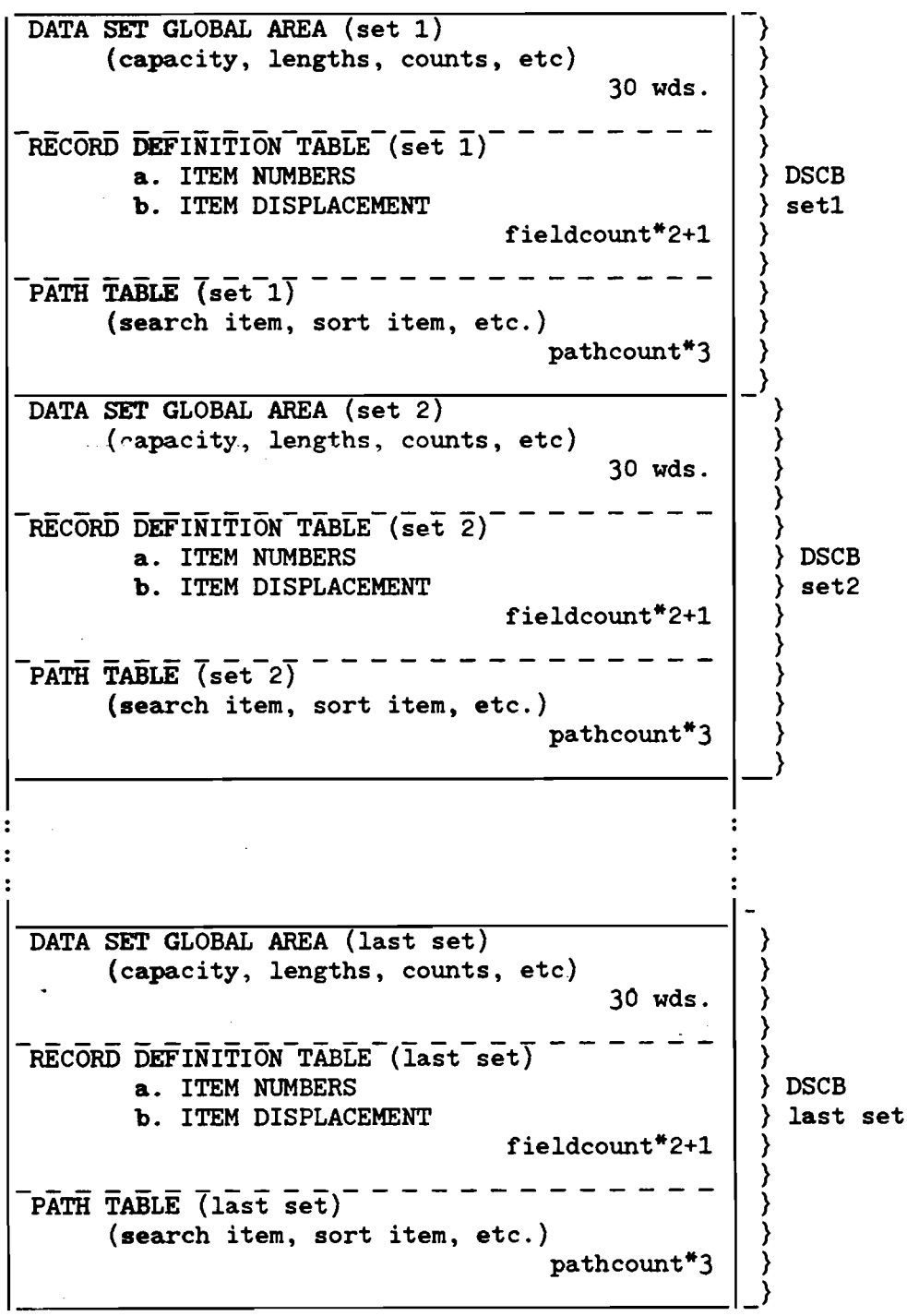
Data-set-type is one of the following: A, M or D.

DSCB-pointer: is a pointer to the Data Set Control Block. This pointer is word offset from record #0. The DSCB is described ahead.

The maximum size for this table is 12*199 = 2388 words.

NOTES: The reserved-1 and reserved-2 fields are the 'old' level numbers for the read and write access respectively. Since this concept no longer applies, the values are set to zero.

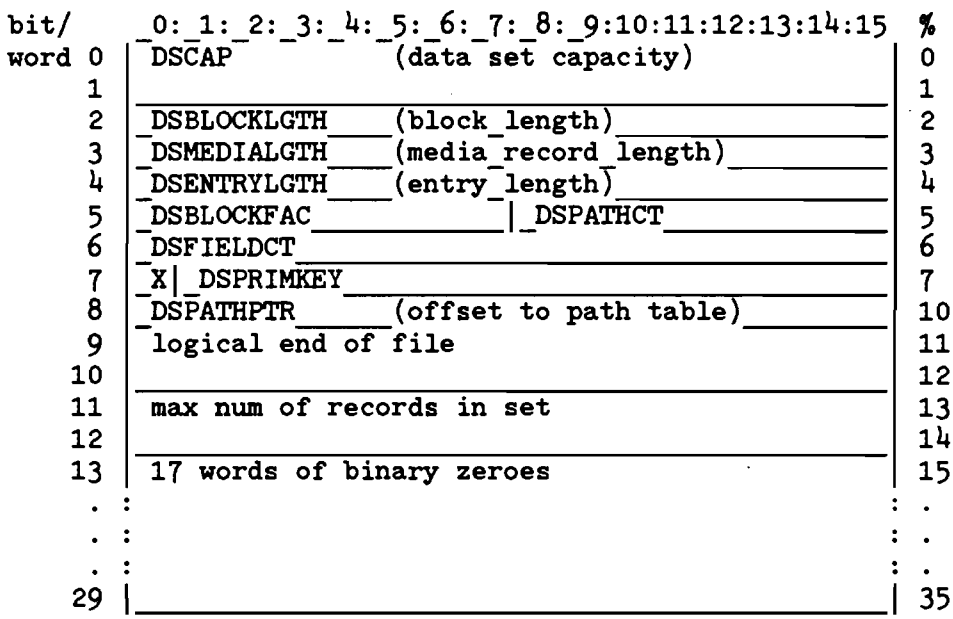
DATA_SET_CONTROL_BLOCKS (DSCB)- General Layout



The DSCBs follow the SET TABLE in the Root File.
There is one DSCB for each data set defined. The function of the DSCB is to define each data set within the data base.

Root File Next Record

DATA SET CONTROL BLOCK (GLOBAL AREA)



DSCAP - data set capacity as reported by the SCHEMA processor.

DSBLOCKLGTH - data set block length including the bit map overhead.

DSMEDIALGTH - data set media record length (remember that this length includes the pointer overhead)

DSENTRYLGTH - data set entry length.

DSBLOCKFAC - data set blocking factor.

DSPATHCT - data set path count. This is the number of paths that are specified for the data set.

DSFIELDCT - data set field count. This is the number of fields specified for the data set.

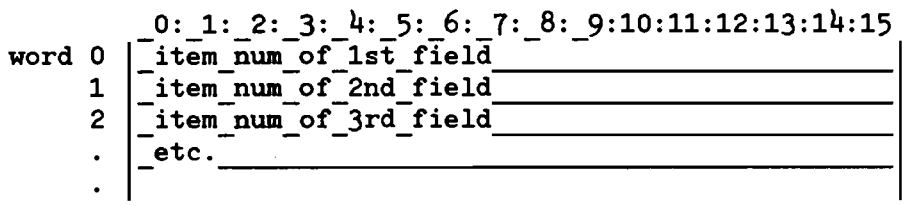
X-DSKEYTYPE - data set key type. If DSKEYTYPE = TRUE then the key is hashed.

DSPRIMKEY - data set primary path or key.
For master data sets, this is the field number of the search item.
For detail data sets, this is the path number of the primary path.

DSPATHPTR - data set path table pointer. Word offset to the data set path table which contains an entry for each path defined. It points to path 0th entry in the table, so to get to the first entry the pointer should be incremented by the length of the entry (which is currently 3 words).

Root File- Next Record

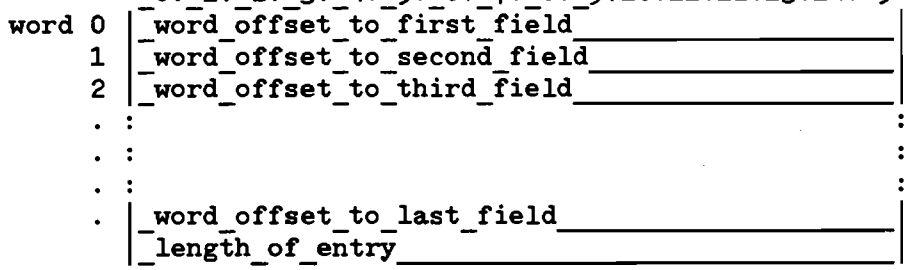
DATA_SET_CONTROL_BLOCK
ITEM_NUMBERS



The Item Numbers Table follows the Global Area of the DSCB. The size of this table (in words) is equal to the number of items in the given data set.

Root File- Next Record

DATA_SET_CONTROL_BLOCK
(RECORD_DEFINITION-ITEM_DISPLACEMENT)



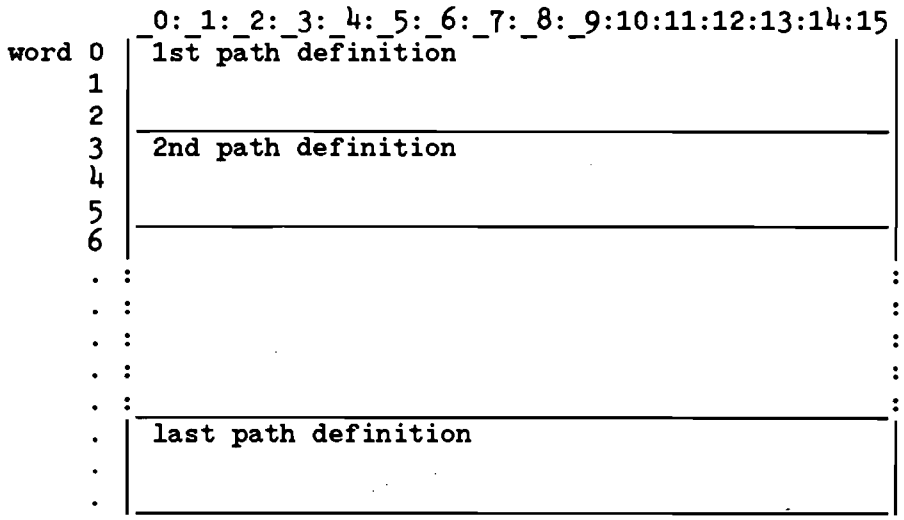
This table immediately follows the Item Numbers Table.

The word offset points to the starting location of the field within the media record. Remember that the media record includes the pointer overhead so this offset varies for master and detail data sets: if a master data set has only one path, the word offset for the first field is 10, since there are 10 words of overhead--5 words for the synonym chain pointers and 5 words for the data set chain head that it would be pointing to. On a detail data set with one path, the overhead is only 4 words.

The 'length-of-entry' field is the same as the media record length.

Root File- Next Record

DATA_SET_CONTROL_BLOCK (PATH_TABLE)



There are 3 words (6 bytes) for each path definition.
The PATH TABLE for master data sets has a different layout from the PATH TABLE for detail data sets.

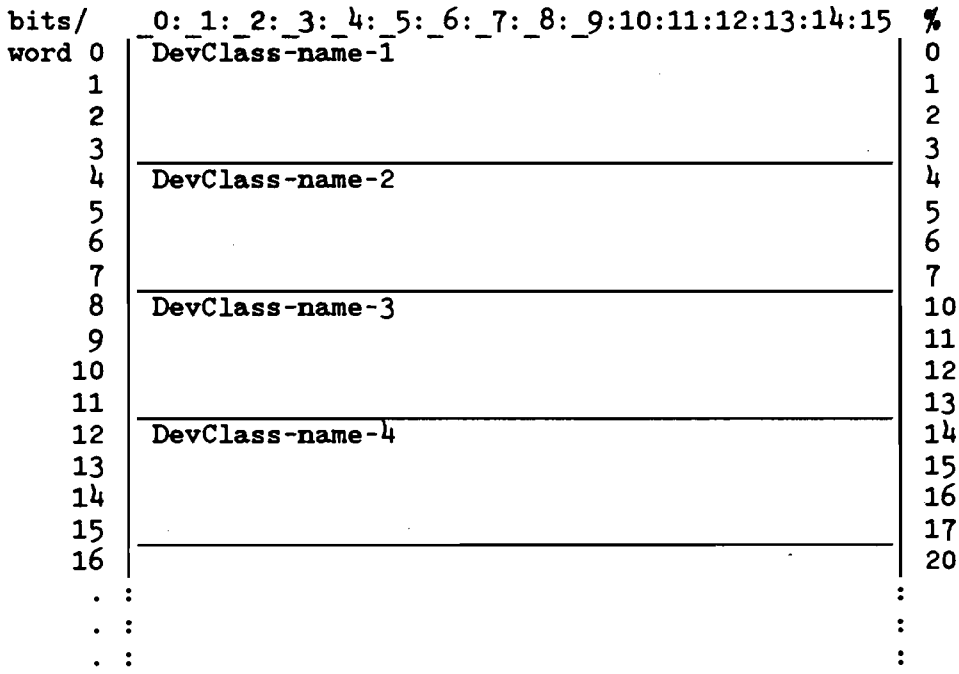
Master sets:

Byte	Description
1-2	item number of the search item in the related detail set.
3-4	item number of the sort item in the related detail set.
5	set number of the related detail data set.
6	path number of the corresponding path in the related detail data set.

Detail sets:

Byte	Description
1-2	field number of the search item.
3-4	field number of the sort item.
5	set number of the related master data set.
6	path number of the corresponding path in the related master data set.

Root File- Next Record



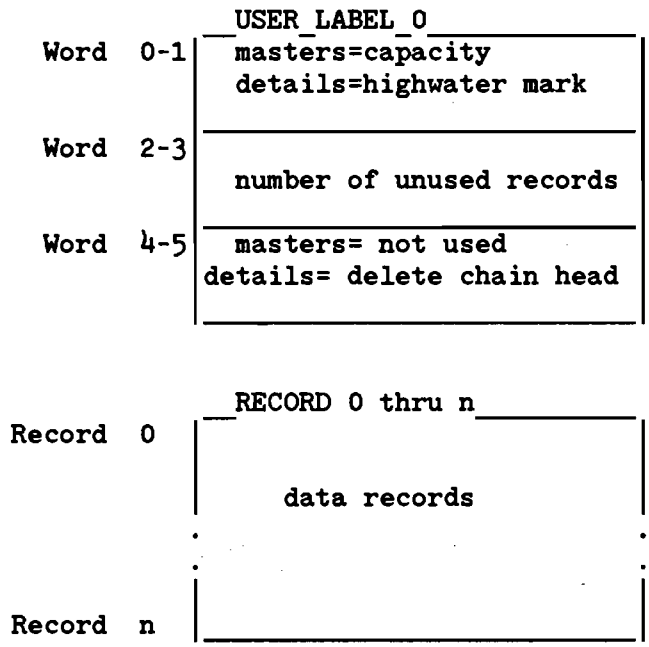
Device Class Table follows the DSCBs.

Each entry is 4 words long, and contains the device class name which is optionally specified for a data set by the user. For data sets without user specified device class names, the entries will be filled with blanks.

The length of the table depends on the number of data sets defined in the schema. The relative position of a device class entry depends on its relative position in the schema.

The maximum size for this table is 4*199 = 796 words.

GENERAL DATA SET LAYOUT



Data Set User Label 0

Word 0-1: Record name of the highest readable record. For Masters, this is the highest record in the set (i.e. *Capacity*). For Details, this is the greatest number of records that have been written to the set thus far. For example, if there is room in the Detail data set for 100 records and 75 were written last week when the data set was loaded with DBLOAD, and yesterday 15 records were deleted from the data set, the *High Water Mark* is equal to a value of '75'.

Word 2-3: Number of unused records in the data set. This field is incremented when a record is deleted and decremented when a record is added. To determine the current number of entries used in the set subtract Word 2-3 (unused count) from Word 0-1 (capacity).

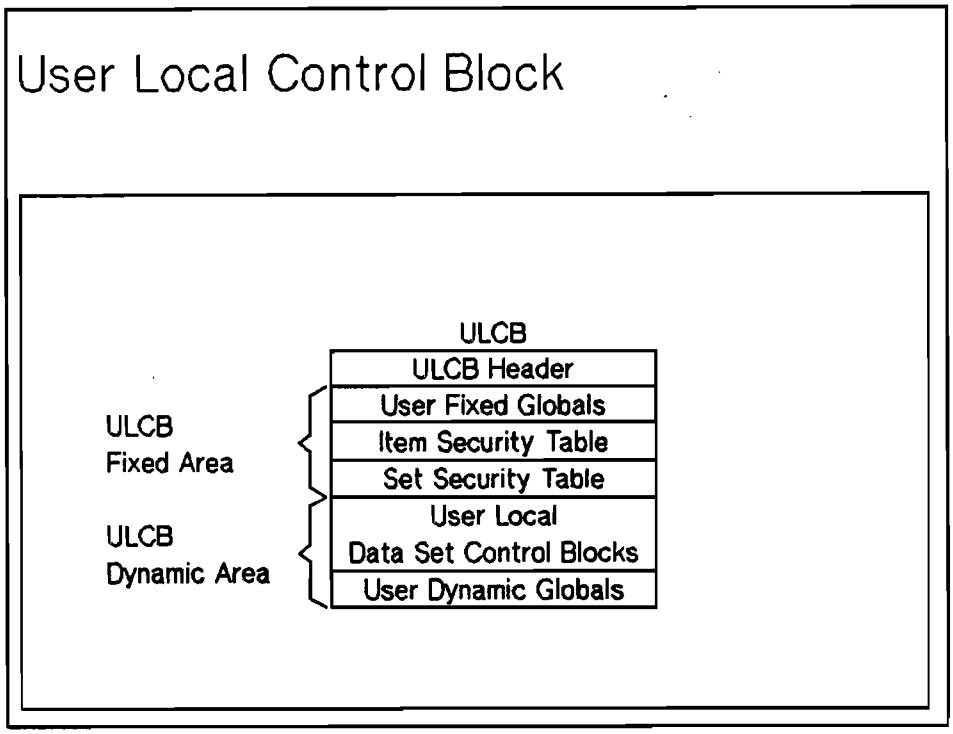
Word 4-5: The delete chain head for Details. This points to the record most recently deleted or contains a value of zero if no records have been deleted. This field is not used in Master data sets.

Data Set Records

The data in the data set records is arranged according to the Media records.

□ ULCB Layout

□ Notes



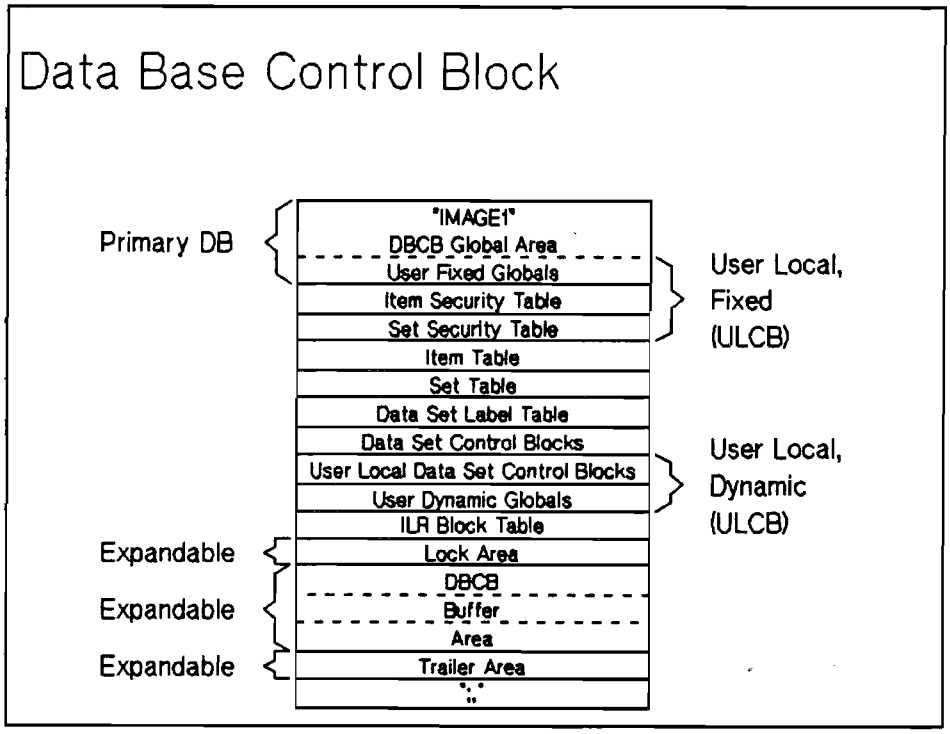
ULCB Header

- Segment type ID: "IMAGE 2".
- DST # of ULCB.
- Base ID (w)
  Open count.
  DBCB DST #.

NOTE: Other portions of the ULCB will be discussed with the DBCB layout.

□ ULCB Layout

□ Notes



Primary DB

- Contains pointers to all portions of the data base.
- Contains global flags set by calls.

User Local Fixed

- Values which differ from user to user but are fixed over time for any one user.

User Local Dynamic

- Values which differ from user to user and differ over time for any one user.

Expandable Areas

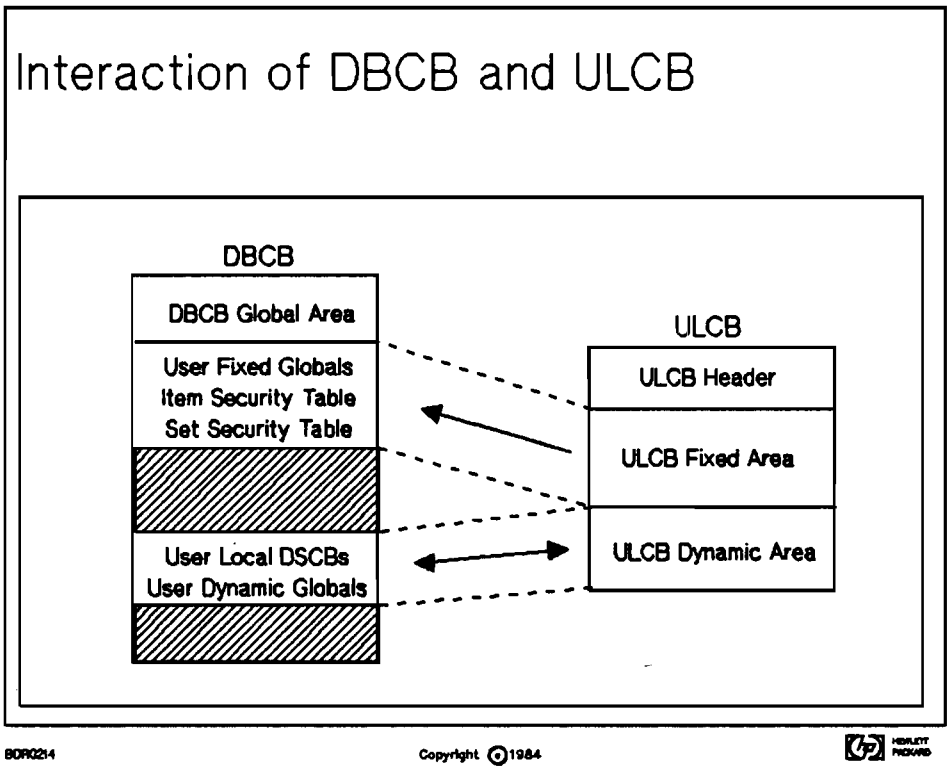
- May dynamically change in size once the data base is open.

Size

- Maximum of 31k words.

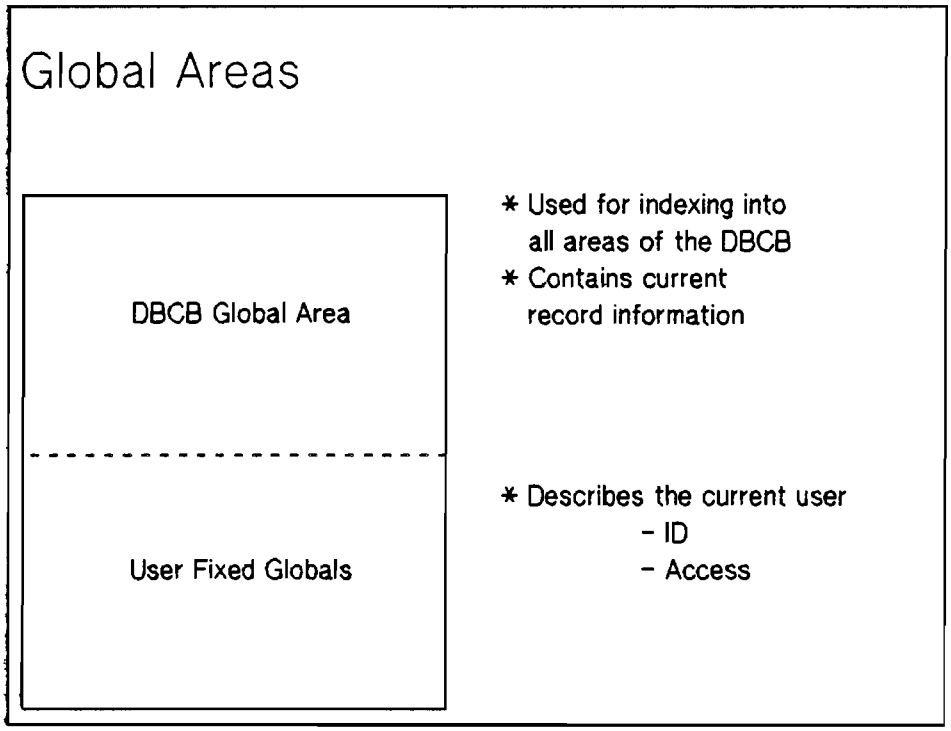
☐ ULCB Layout

☐ Notes



□ DBCB Layout

□ Notes



DBCB Global Area

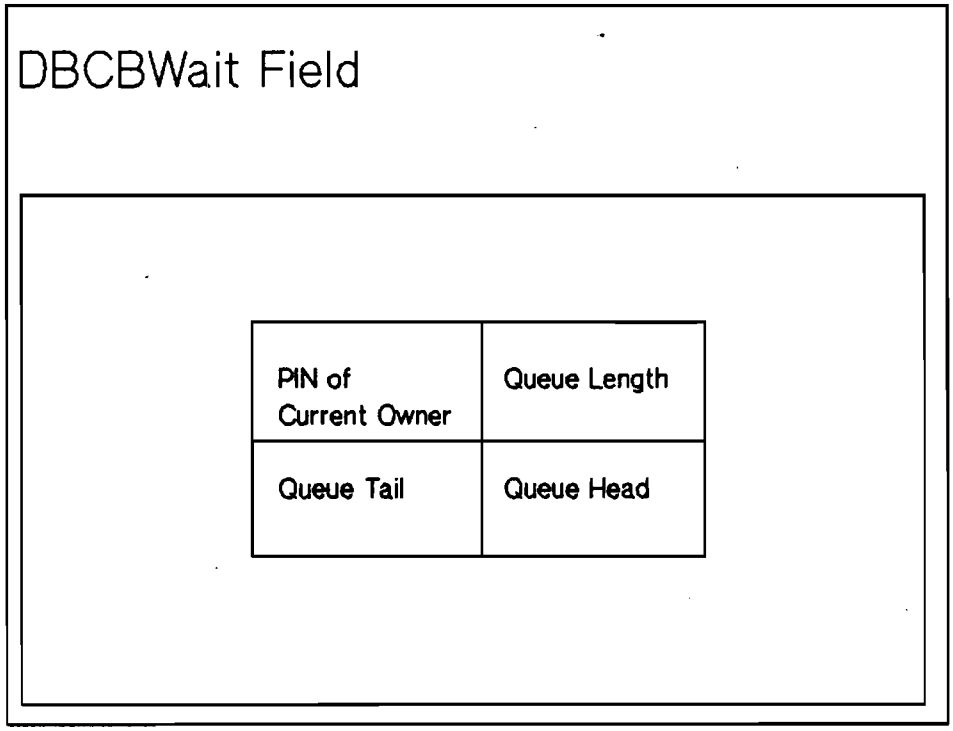
- Segment type ID: "IMAGE 1".
- Pointers to beginning of...
  - Each Table
  - DSCB Area
  - ULDSCB Area
  - User Dynamic Global Area
  - Lock Area
  - Buffer Area
  - Trailer Area
- DST # of the ULCB.
- Item/set map tables (used to hash item/set name to locate item/set in item/set table).
- Counts and lengths.
  - Data Buffers
  - Trailer Area
- Available buffer queue.
  - First Available Buffer
  - Last Available Buffer
- DBCB wait field.
  - First Waiter
  - Last Waiter
- Scratch pointers.
  - Current data item and set informatiom
  - Current DSCB & ULDSCB
  - Current Path
  - Current Buffer

User Fixed Globals

- Current user ID.
- Data set a-options.
- Update, Put and Delete flags.
- File number of root file.

□ DBCB Layout

□ Notes



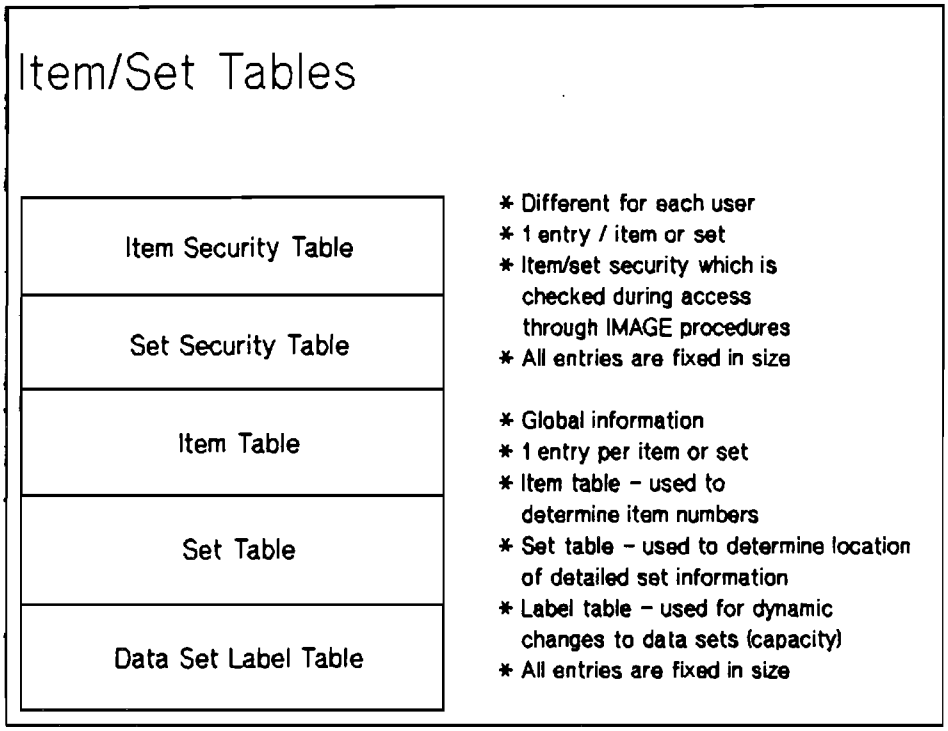
Used to lock the DBCB.

Must gain control of this before executing most IMAGE intrinsics.

Contained in DBCB global area.

□ DBCB Layout

□ Notes



Item Security Table
- 1 read byte per item
- 1 write byte per item

Set Security Table
- 1 read byte per set
- 1 write byte per set

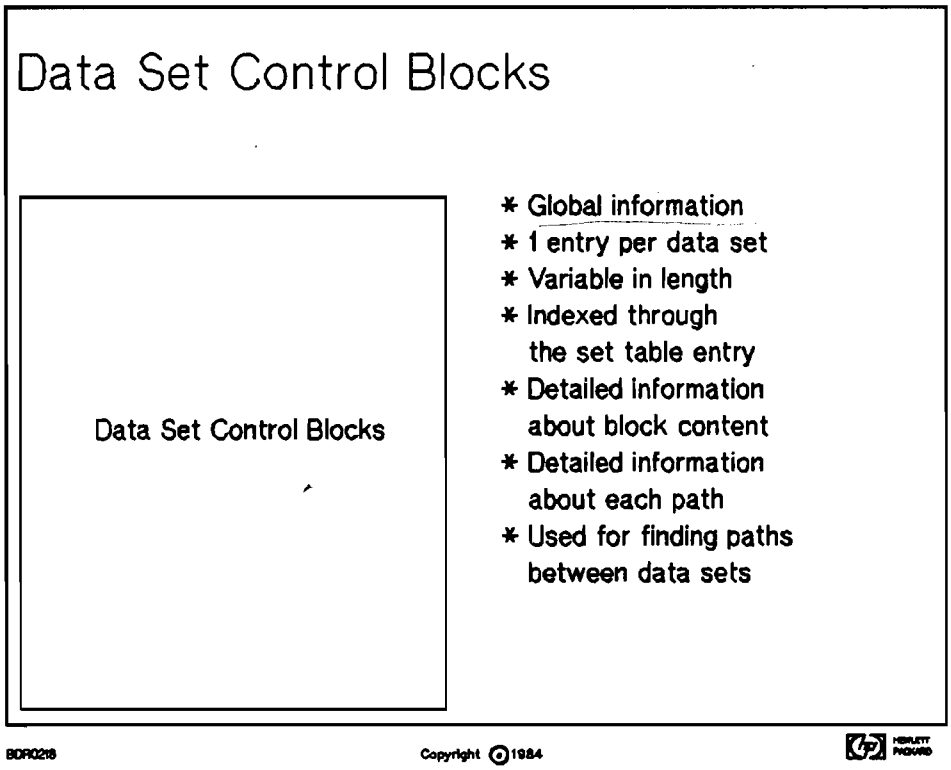
Item Table
- Data item name
- Data item type
- Subitem count
- Subitem length

Set Table
- Data set name
- Data set type
- DSCB pointer
- ULDSCB pointer

Label Table
- Data set EOF
- Free record count
- Free chain head (D)

□ DBCB Layout

□ Notes

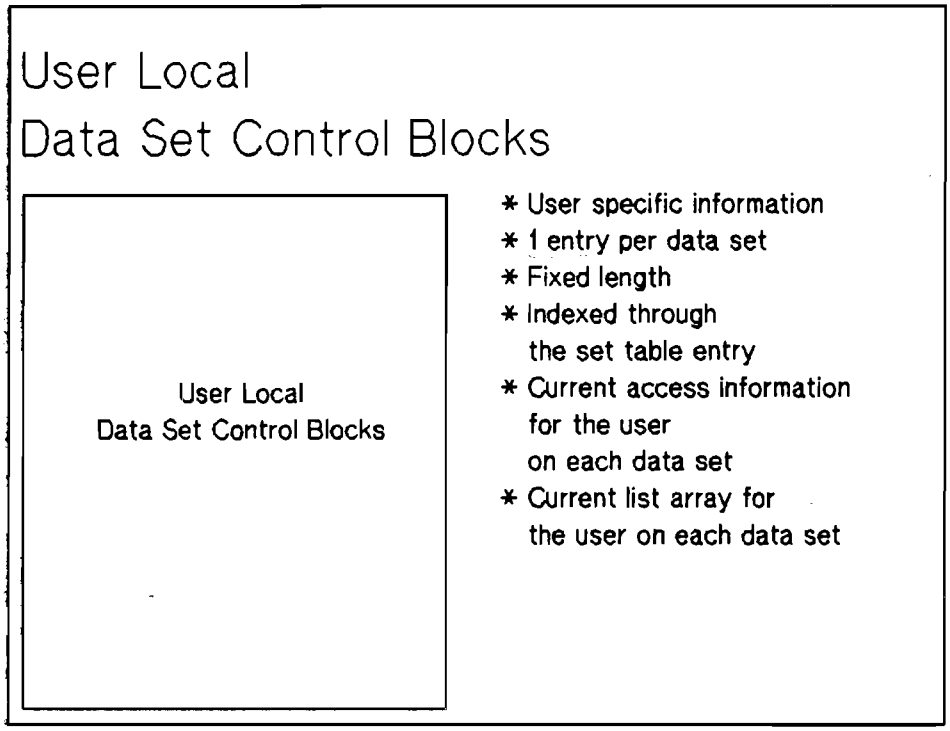


Data Set Control Block

- Header
  - Capacity
  - Block, entry size
  - Media record size
  - Block factor
  - # items per record
  - # paths
  - Primary path
- Record definition
  - Item # of each item
  - Media record offset of each field
- Path table (4 byte entry/path)
  - Master
    - * Item # of search item in related detail
    - * Item # of sort item in related detail
    - * Set # of related detail
    - * Path # of path in related detail
  - Detail
    - * Field # of search item
    - * Field # of sort item
    - * Set # of related master.
    - * Path # of path in related master

□ DBCB Layout

□ Notes

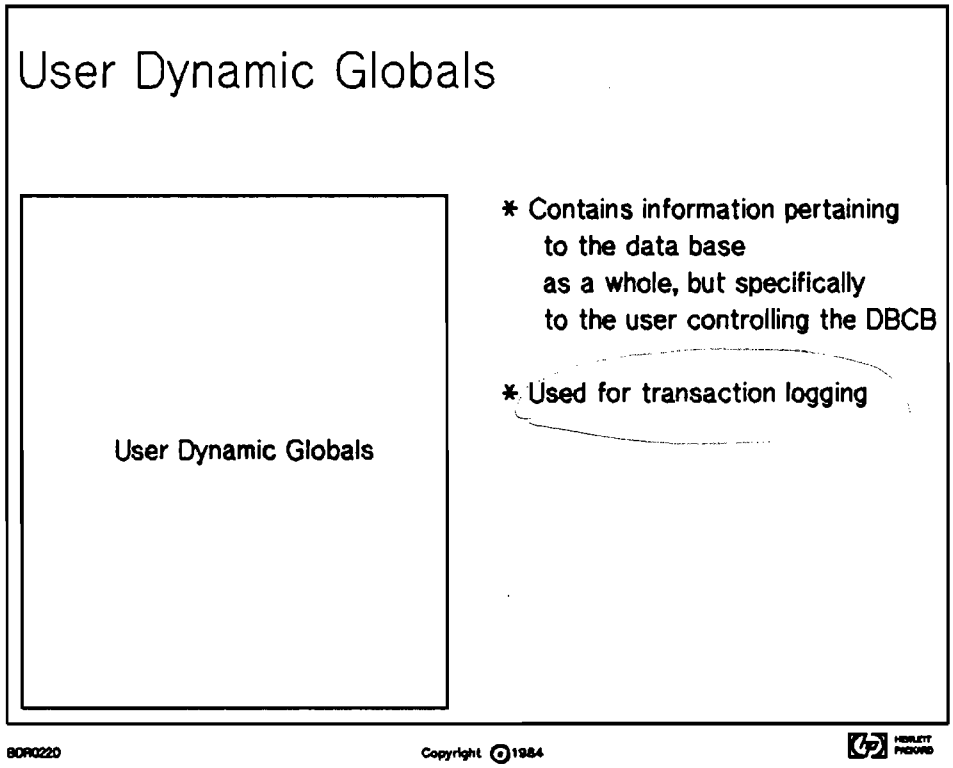


ULDSCB

- Header
  - Current list name
  - Chain length
  - Backward pointer
  - Forward pointer
  - Current path
  - Last procedure
  - Last mode
  - Data set file number
- Current list array
  - Current list by field #

□ DBCB Layout

□ Notes



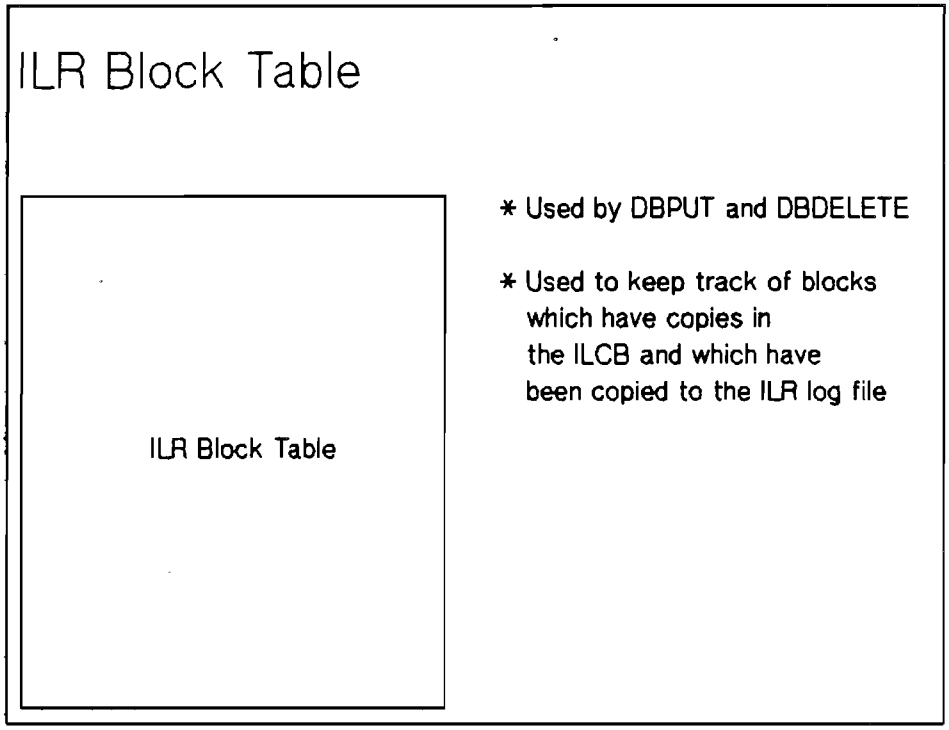
Contains:

Transaction Count

Transaction Number

□ DBCB Layout

□ Notes



The table is divided into 2 areas.

1. Index Area
   - Free space pointer - points to next 4 available words in index area.
   - Set pointers - one for each set in data base, points to first four word entry for that data set.

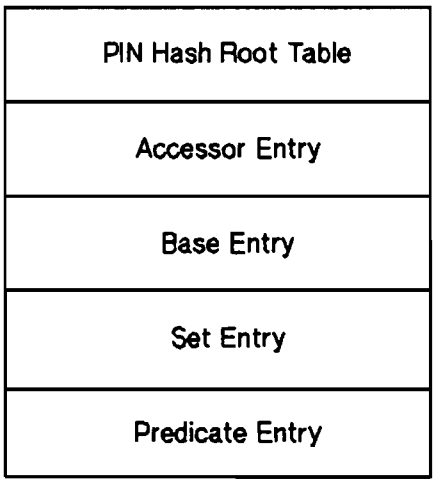
2. Record Area
   - Four word entries - each assigned entry contains:
     Pointer to next entry for particular set.
     Flushed flag (tells whether block has been flushed to ILR log file).
     Data set block number (2 words).

□ DBCB Layout

□ Notes

keeps track of what locks a PIN holds

DBCB Lock Area



Major Areas

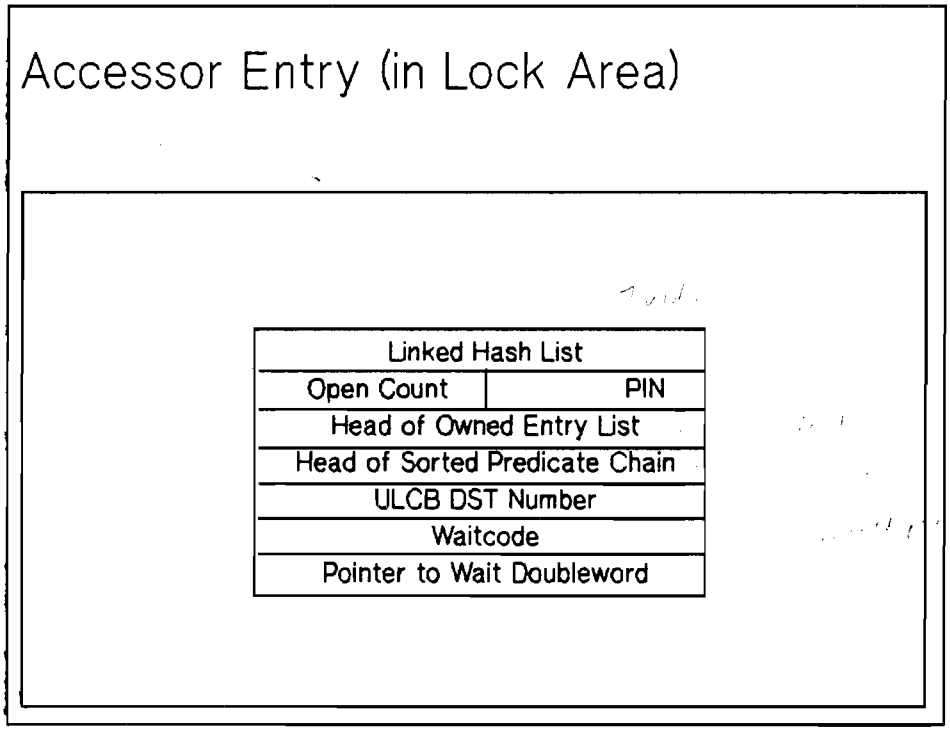
1. PIN Hash Root Table
   Created by DBOPEN
2. Accessor Entry
   Created by DBOPEN
3. Base Entry
   Created by DBOPEN
4. Set Entry
   Created by DBLOCK
5. Predicate Entry
   Created by DBLOCK
   Released by DBUNLOCK

BDR0222

HEWLETT PACKARD

□ DBCB Layout

□ Notes



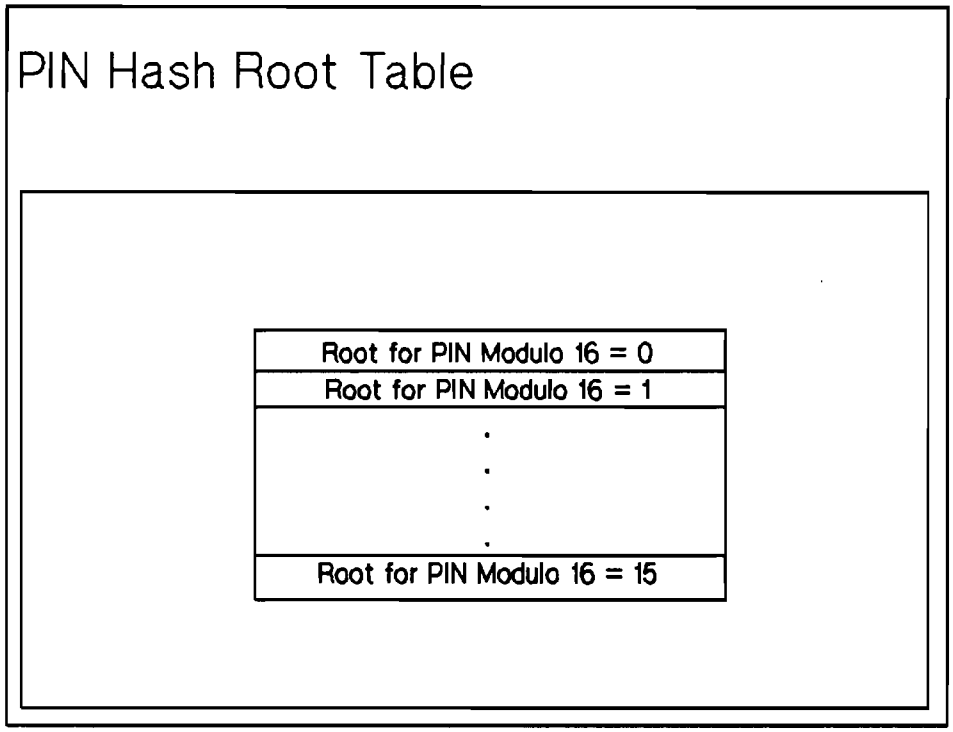
Created by DBOPEN.

1 entry per DBOPEN.

Identifies who is using the data base.

□ DBCB Layout

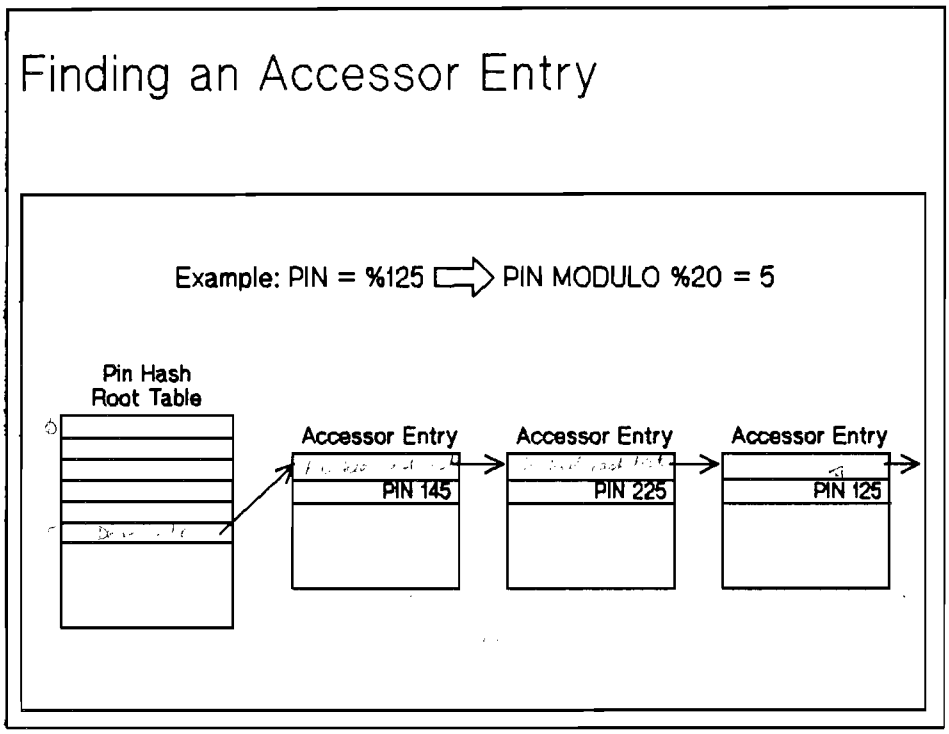
□ Notes



Each accessor entry is attached to a linked list whose root is in this table.

□ DBCB Layout

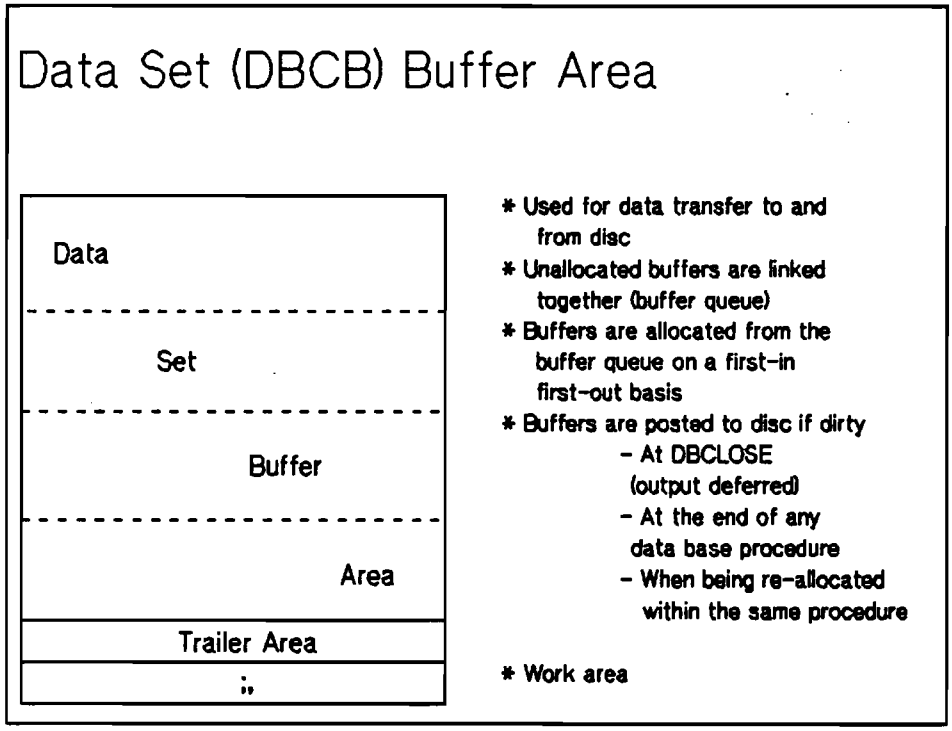
□ Notes



Module 2.30

□ DBCB Layout

□ Notes



Data Buffer

- Header
  - Previous Buffer in Queue
  - Next Buffer in Queue
  - Block #
  - Set #
  - Media Record Length
  - Block Factor
  - Busy Flag (tells whether buffer is in use)
  - Dirty Flag (tells whether buffer has been modified)
- Buffer
  - Size of Data Block

Trailer Area (128w)

- Used for data transfer to and from stack.

□ DBCB Layout

□ Notes

Maximum Buffers

MxB = [30400 - 12i - 43s - 2(TFC + TPC)] / (BL + 9)

where:

i	is the Item Name Count
s	is the Data Set Count
TFC	is the Total Field Count
TPC	is the Total Path Count
BL	is the Buffer Length

BDR0227 HEWLETT PACKARD

Data Set Buffers

- Each buffer is as large as the largest data block + 10w.
- Default allocation.

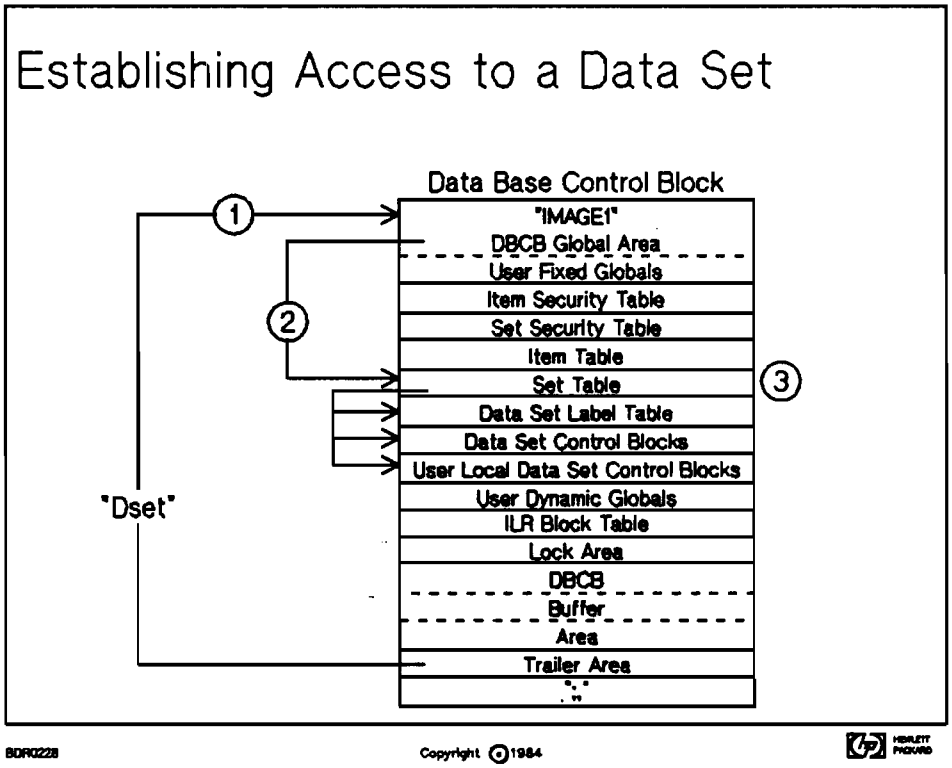
  1. Calculates the maximum number of paths into any given data set + 3.
  2. Allocates (maxpaths + 3) or 8 whichever is greater.
  3. Allocates 1 buffer for each additional 2 users (up to 20 users).

- A single user never needs more than 4 buffers for any given procedure.
- Record level locking may cause buffer thrashing.
- Default allocation can be overriden by DBUTIL set command.
- Number of buffers should be kept constant over a range of users to eliminate DBCB contractions/expansions during DBOPEN/DBCLOSE.

Module 2.32

□ DBCB Layout

□ Notes



Establishing Access to a Data Set

1. Index into the global area to locate the set table with the set map table.

2. Scan the set table and determine the set #, ULDSCB, DSCB, and set label locations.

3. Set indices to the . . .

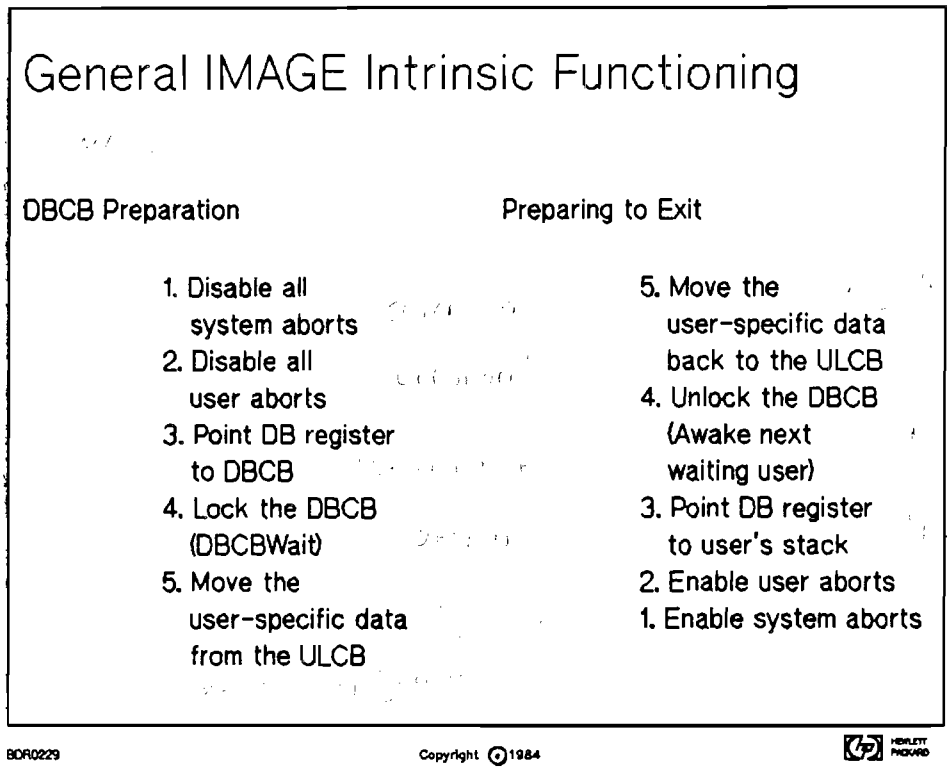
Data Set Label - contains capacity and free space chain head.
- Space Chain Head

DSCB - contains the set format.
- Set Definition
- Record Definition
- Path Definition

ULDSCB - contains user data set pointers.
- Current, Forward, and Backward Pointers
- Data Set Filenumber
- Current List

□ DBCB Layout

□ Notes



Requested function is done after DBCB preparation and before preparing to exit.

□ DBCB Layout

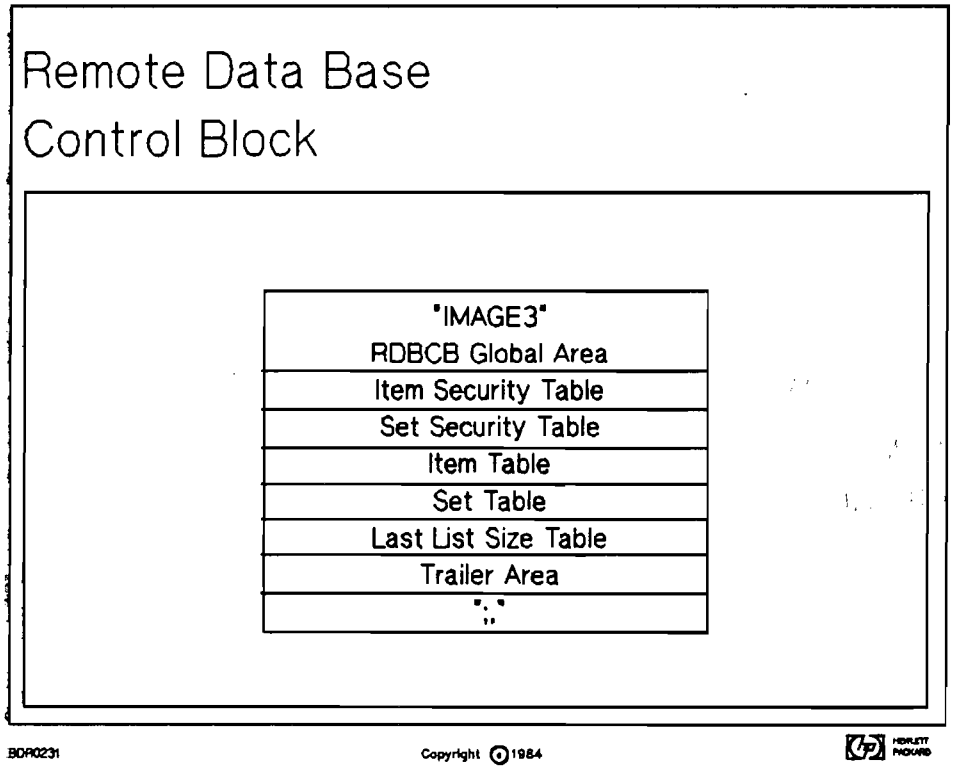
□ Notes

DBDRIVER

* Runs in non-priviledged mode
* Allows interactive calls to IMAGE data bases
* Returns time (in milliseconds) to complete call
  elapsed time (not CPU)
* Allows parameters to be set !
* Allows displaying of parameters ?
* Allows all DBMS intrinsic calls to be executed

□ RDBCB Layout

□ Notes



1 per user opening a remote data base.

Last List Size Table contains the entry size for the Current List of each data set (Current List is contained in that user's ULCB).

QUIZ

Specify where in the DBCB the following information is stored.

1. DBCB wait field In the DBCB global

2. Accessor entry in the lock area

3. Update flag for a given user in fixed global

4. Read/Write byte for a given item for a given user Item/set access table

5. Current path for a given set for a given user UL DSCB

6. Capacity of a data set DBCB set table.

7. Type of an item Item table

8. Free chain head for a detail data set [Use labeled file] Data set label table

9. Data block (media record) Buffer area

10. DSCB pointer for a given set Set table

11. Explain the interaction of the ULCB and DBCB. data from ULCB --> DBCB at beginning [illegible], once the interaction is complete and user unlocks,

ULCB DBCB

All [illegible] the data [illegible] has been sent back to ULCB.

```
00000 0  << Compiled with maint. file:    M09M215B      B.04.00           >>
00000 0  $CONTROL SUBPROGRAM,MAP,LIST,SOURCE,ADR,CODE
00000 0
00000 0  $INCLUDE DBINCLD
00000 0  $CONTROL SUBPROGRAM,MAP,LIST,SOURCE,ADR,CODE
00000 0  <<*****************************************************************>>
00000 0  <<                                                                 >>
00000 0  <<              IMAGE/3000  --  DATA BASE MANAGEMENT SYSTEM        >>
00000 0  <<                     LIBRARY PROCEDURES (INTRINSICS)             >>
00000 0  <<                                                                 >>
00000 0  <<*****************************************************************>>
00000 0
00000 0  $COPYRIGHT                                                    &
00000 0  $                                                             &
00000 0  $"        (C)    COPYRIGHT HEWLETT-PACKARD.     1978         ",&
00000 0  $" This program may be used with one computer system at a time ",&
00000 0  $" and shall not otherwise be recorded, transmitted or stored  ",&
00000 0  $" in a retrieval system. Copying or other reproduction of this",&
00000 0  $" program except for archival purposes is prohibited without  ",&
00000 0  $" the prior written consent of the Hewlett-Packard Company.   "
00000 0
00000 0  << Compiled with maint. file:    MINCLD        B.04.Y1          >>
00000 0  $TITLE "32215B, IMAGE/3000:  INTRINSIC OUTER BLOCK AND GLOBALS"
00000 0  BEGIN
00000 1
00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
00000 1  << THE FOLLOWING DECLARATIONS FORM THE "OUTER BLOCK" FOR ALL IMAGE    >>
00000 1  <<   INTRINSICS AND PROCEDURES.  INCLUDED ARE NUMEROUS COMMONLY USED  >>
00000 1  <<   EQUATES AND DEFINES, PLUS ALL IDENTIFIERS USED FOR SYMBOLIC      >>
00000 1  <<   ACCESS TO THE DATA BASE CONTROL BLOCK.  THE DBCB IS RESIDENT IN  >>
00000 1  <<   A PRIVILEGED EXTRA DATA SEGMENT; IT IS ALLOCATED AND INITIALIZED >>
00000 1  <<   BY DBOPEN, WHICH READS THE CONTENTS OF THE ROOT FILE INTO THE    >>
00000 1  <<   EXTRA DATA SEGMENT (XDS), AND THEN MODIFIES CERTAIN TABLES AND   >>
00000 1  <<   VARIABLES ACCORDING TO THE ACCESS BEING GRANTED TO THE CALLER.   >>
00000 1  << NONE OF THE FOLLOWING GLOBAL (DB-RELATIVE) VARIABLE DECLARATIONS   >>
00000 1  <<   ACTUALLY ALLOCATES STORAGE.  INSTEAD, ALL MERELY ALLOW ANY OF    >>
00000 1  <<   THE IMAGE PROCEDURES TO ACCESS SYMBOLICALLY ELEMENTS OF THE      >>
00000 1  <<   DBCB.  THIS CAN BE DONE, HOWEVER, ONLY WHEN THE DB-REGISTER IS   >>
00000 1  <<   POINTING TO THE XDS CONTAINING THE DBCB.  ANY USE OF THESE       >>
00000 1  <<   SYMBOLS WITH DB POINTING TO THE USER'S STACK IS INVALID.         >>
00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
```

DB+230 ptr hash table / hash base

MPE V/P — IMAGE

B 04 02 since


```
00000 1  <<                LAYOUT OF DATA BASE CONTROL BLOCK
00000 1
00000 1
00000 1                   **   ****************************
00000 1                    *   *                          *
00000 1                    *   *     DBCB GLOBAL AREA     *
00000 1     PRIMARY DB ***     *                          *
00000 1                    *   *  *  *  *  *  *  *  *  *  *   **
00000 1                    *   *    USER FIXED GLOBALS    *    *
00000 1                   **   ****************************    *
00000 1                        *   ITEM SECURITY TABLE    *    *** USER LOCAL,
00000 1                        ****************************    *     FIXED
00000 1                        *    SET SECURITY TABLE    *    *
00000 1                        ****************************   **
00000 1                        *                          *
00000 1                        *                          *
00000 1                        *        ITEM TABLE        *
00000 1                        *                          *
00000 1                        *                          *
00000 1                        *                          *
00000 1                        ****************************
00000 1                        *                          *
00000 1                        *                          *
00000 1                        *         SET TABLE        *
00000 1                        *                          *
00000 1                        *                          *
00000 1                        *                          *
00000 1                        ****************************
00000 1                        *                          *
00000 1                        *   DATA SET LABEL TABLE   *
00000 1                        *                          *
00000 1                        ****************************
00000 1                        *                          *
00000 1                        *         DATA SET         *
00000 1                        *      CONTROL BLOCKS      *
00000 1                        *         (DSCB'S)         *
00000 1                        *                          *
00000 1                        ****************************   **
00000 1                        *                          *    *
00000 1                        *        USER LOCAL        *    *
00000 1                        *         DATA SET         *    *
00000 1                        *      CONTROL BLOCKS      *    *** USER LOCAL,
00000 1                        *        (ULDSCB'S)        *    *     DYNAMIC
00000 1                        *                          *    *
00000 1                        ****************************    *
00000 1                        *   USER DYNAMIC GLOBALS   *    *
00000 1                        ****************************   **
00000 1
00000 1
00000 1
00000 1                              (CONTINUED)
```

```
 89     00000 1                    ****************************
 90     00000 1                    *                          *
 91     00000 1                    *  BLOCK TABLE FOR RECOVERY *
 92     00000 1                    *                          *
 93     00000 1                    ****************************   **
 94     00000 1                    *                          *    *
 95     00000 1                    *                          *    *
 96     00000 1                    *        LOCK AREA         *    *** EXPANDABLE
 97     00000 1                    *                          *    *
 98     00000 1                    *                          *    *
 99     00000 1                    ****************************   ** **
100     00000 1                    *                          *      *
101     00000 1                    *                          *      *
102     00000 1                    *          DATA            *      *
103     00000 1                    *                          *      *
104     00000 1                    *                          *      *
105     00000 1                    ****                  ******      *
106     00000 1                    *                          *      *
107     00000 1                    *                          *      *
108     00000 1                    *          SET             *      *
109     00000 1                    *                          *      *
110     00000 1                    *                          *      *
111     00000 1                    ****                  ******      *** EXPANDABLE
112     00000 1                    *                          *      *
113     00000 1                    *                          *      *
114     00000 1                    *         BUFFER           *      *
115     00000 1                    *                          *      *
116     00000 1                    *                          *      *
117     00000 1                    ****                  ******      *
118     00000 1                    *                          *      *
119     00000 1                    *                          *      *
120     00000 1                    *          AREA            *      *
121     00000 1                    *                          *      *
122     00000 1                    *                          *      *
123     00000 1                    ****************************   ** **
124     00000 1                    *                          *    *
125     00000 1                    *       TRAILER AREA       *    *** EXPANDABLE
126     00000 1                    *                          *    *
127     00000 1                    ****************************   **
128     00000 1                    *          ";"             *
129     00000 1                    ****************************
130     00000 1
131     00000 1
132     00000 1                                                                  >>
```



```
134     00000 1  <<                 ****************************
135     00000 1                     *                          *
136     00000 1                     *     DBCB GLOBAL AREA     *
137     00000 1                     *                          *
138     00000 1                     *  *  *  *  *  *  *  *     *
139     00000 1                     *    USER FIXED GLOBALS    *
140     00000 1                     ****************************                  >>
141     00000 1
142     00000 1
143     00000 1
144     00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
145     00000 1  << BEGINNING OF DATA BASE GLOBAL AREA.  THESE ELEMENTS ARE PRESENT IN >>
146     00000 1  <<   THE FIRST PART OF THE DBCB, AND ARE ACCESSED DIRECTLY.          >>
147     00000 1  <<   THEY INCLUDE, AMONG OTHER THINGS, FIXED POINTERS (INITIALIZED   >>
148     00000 1  <<   BY DBOPEN) TO ALL THE OTHER TABLES AND AREAS.  THE FUZZY       >>
149     00000 1  <<   DISTINCTION BETWEEN TABLE AND AREA IS:                          >>
150     00000 1  <<        A TABLE IS A GROUP OF FIXED LENGTH ENTRIES; THE GLOBAL     >>
151     00000 1  <<           POINTER POINTS TO THE (NONEXISTENT) 0TH ENTRY.          >>
152     00000 1  <<        AN AREA IS A GROUP OF ENTRIES OR ENTITIES, NOT NECESSARILY >>
153     00000 1  <<           FIXED LENGTH; THE GLOBAL POINTER POINTS TO THE ACTUAL   >>
154     00000 1  <<           BEGINNING OF THE AREA.                                  >>
155     00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
156     00000 1
157     00000 1
158     00000 1  ARRAY TAG(0:2)=DB;          << SEGMENT TYPE ID>>
                         DB+000
159     00000 1  INTEGER DBCB'DST;           << DST NUMBER OF THIS XDS (DBCB) >>
                         DB+003
160     00000 1
161     00000 1  << POINTERS TO AREAS WITHIN DBCB >>
162     00000 1
163     00000 1  INTEGER POINTER ITEMSECTBL;
                         DB+004
164     00000 1     << WORD POINTER TO 0TH ITEM'S READ/WRITE BYTES WORD >>
165     00000 1
166     00000 1  INTEGER POINTER DSETSECTBL;
                         DB+005
167     00000 1     << WORD POINTER TO 0TH SET'S READ/WRITE BYTES WORD >>
168     00000 1
169     00000 1  INTEGER POINTER ITEMTBL;
                         DB+006
170     00000 1     << WORD POINTER TO 0TH ENTRY OF ITEM TABLE >>
171     00000 1
172     00000 1  INTEGER POINTER DSETTBL;
                         DB+007
173     00000 1     << WORD POINTER TO 0TH ENTRY OF SET TABLE >>
174     00000 1
175     00000 1  INTEGER POINTER DSLABELTBL;
                         DB+010
176     00000 1     << WORD POINTER TO 0TH 6-WORD DSET LABEL DATA (EOF, ETC.) >>
177     00000 1
178     00000 1  INTEGER POINTER DSCBAREA;
                         DB+011
179     00000 1     << WORD POINTER TO BEGINNING OF FIRST DSCB >>
180     00000 1
181     00000 1  INTEGER POINTER ULDSCBAREA;
                         DB+012
```

```
PAGE 0006   HEWLETT-PACKARD 32215B, IMAGE/3000:  INTRINSIC OUTER BLOCK AND GLOBALS

  182     00000 1       << WORD POINTER TO BEGINNING OF FIRST ULDSCB >>
  183     00000 1
  184     00000 1   INTEGER POINTER USERDYNAMICAREA;
                          DB+013
  185     00000 1       << WORD POINTER TO USER DYNAMIC "GLOBALS" >>
  186     00000 1
  187     00000 1   INTEGER POINTER ILR'BLOCK'TABLE;
                          DB+014
  188     00000 1   << WORD POINTER TO THE IN-USE BLOCK TABLE FOR ILR LOGGING >>
  189     00000 1
  190     00000 1   INTEGER POINTER LOCKAREA;
                          DB+015
  191     00000 1       << WORD POINTER TO BEGINNING OF LOCK BLOCKS (JOHN PAGE AREA) >>
  192     00000 1
  193     00000 1   INTEGER POINTER BUFFERAREA;
                          DB+016
  194     00000 1       << WORD POINTER TO BEGINNING OF BUFFER AREA >>
  195     00000 1
  196     00000 1   INTEGER POINTER TRLR;
                          DB+017
  197     00000 1       << WORD POINTER TO TRAILER (WORK) AREA >>
  198     00000 1
  199     00000 1   INTEGER XDSLGTH;
                          DB+020
  200     00000 1       << TOTAL (CURRENT) LENGTH OF DBCB'S EXTRA DATA SEGMENT >>
  201     00000 1
  202     00000 1
  203     00000 1   << DATA BASE IDENTIFICATION >>
  204     00000 1
  205     00000 1   INTEGER SDBCB'INDEX;              << Where SDBCB has the DST#.   >>
                          DB+021
  206     00000 1   INTEGER DSNUM;                 << DS LINE # (0 MEANS LOCAL BASE) >>
                          DB+022
  207     00000 1   INTEGER REMDBNAME'LENGTH;     << LENGTH OF REMOTE DATA BASE NAME >>
                          DB+023
  208     00000 1   INTEGER ARRAY REMDBNAME(0:19)=DB;  << 20-WORD REMOTE DB NAME >>
                          DB+024
  209     00000 1
  210     00000 1   DOUBLE ARRAY DBNAME(*) = REMDBNAME; << DATA BASE NAME (6 BYTES)>>
                          DB+024
  211     00000 1   DOUBLE ARRAY DBGROUP(*)= REMDBNAME+3; << DATA BASE'S GROUP NAME >>
                          DB+027
  212     00000 1   DOUBLE ARRAY DBACCT(*) = REMDBNAME+7; << DATA BASE'S ACCOUNT NAME >>
                          DB+033
  213     00000 1
  214     00000 1
  215     00000 1   << USER LOCAL CONTROL BLOCK (ULCB) DESCRIPTORS >>
  216     00000 1
  217     00000 1   INTEGER ULFIXED'OFFSET;
                          DB+050
  218     00000 1   INTEGER ULFIXED'LENGTH;
                          DB+051
  219     00000 1   INTEGER ULDYNAMIC'OFFSET;
                          DB+052
  220     00000 1   INTEGER ULDYNAMIC'LENGTH;
                          DB+053
  221     00000 1

PAGE 0006   HEWLETT-PACKARD 32215B, IMAGE/3000:  INTRINSIC OUTER BLOCK AND GLOBALS

  222     00000 1
  223     00000 1   << COUNTS, LENGTHS, FLAGS >>
  224     00000 1
  225     00000 1   LOGICAL  PREMPE5;                 << See lock area for detail info. >>
                          DB+054
  226     00000 1   LOGICAL LOGGING;                      << DATA BASE IS BEING LOGGED >>
                          DB+055
  227     00000 1   INTEGER ARRAY   LOGID(0:3) = DB;         << LOG IDENTIFIER >>
                          DB+056
  228     00000 1   INTEGER ARRAY   LOGPASS(0:3) = DB;       << LOG ID PASSWORD >>
                          DB+062
  229     00000 1   INTEGER         LOGFLAGS;
                          DB+066
  230     00000 1   INTEGER ARRAY   STOREDATE(0:2) = DB;     << DATE, TIME OF DBSTORE >>
                          DB+067
  231     00000 1   INTEGER ITEMCT;                 << NUMBER OF DATA ITEMS >>
                          DB+072
  232     00000 1   INTEGER DSETCT;                 << NUMBER OF DATA SETS >>
                          DB+073
  233     00000 1   INTEGER BUFFERCOUNT;            << CURRENT NUMBER OF DATA BUFFERS >>
                          DB+074
  234     00000 1   INTEGER QUEUECOUNT;           << Initially 0, ever <=BUFFERCOUNT >>
                          DB+075
  235     00000 1   INTEGER QUEUEHEAD;            << FIRST IN AVAILABLE BUFFER QUEUE >>
                          DB+076
  236     00000 1   INTEGER QUEUETAIL;             << LAST IN AVAILABLE BUFFER QUEUE >>
                          DB+077
  237     00000 1   INTEGER SHORT'SET;            << Used by FINDRECORD >>
                          DB+100
  238     00000 1   INTEGER SHORT'CUT;            << Used by FINDRECORD >>
                          DB+101
  239     00000 1   INTEGER BUFFLGTH;               << WD LGTH OF EACH BUFFER (DATA ONLY)>>
                          DB+102
  240     00000 1   INTEGER TRLRLGTH;               << WORD LENGTH OF TRAILER AREA >>
                          DB+103
  241     00000 1   INTEGER BUFFERENTRYLENGTH;      << WORD LGTH OF EACH ENTIRE BUFFER >>
                          DB+104
  242     00000 1                                   <<     ( HEADER AND DATA )          >>
  243     00000 1
  244     00000 1   <<-------------------------------------------------------------------->>
  245     00000 1   << The Global DBCB(*) declarations for ILR are as follows:           >>
  246     00000 1   <<-------------------------------------------------------------------->>
  247     00000 1
  248     00000 1           <<---------------------------------------------------------->>
  249     00000 1           << For DBINFO with MODE = 402 calls:                       >>
  250     00000 1           <<---------------------------------------------------------->>
  251     00000 1
  252     00000 1   integer ILR'DSETNUM,            << Which data set was being touched ? >>
                          DB+105
  253     00000 1           ILR'PCODE,              << Which intrinsic was executing ?    >>
                          DB+106
  254     00000 1           ILR'ON'DATE;            << When was ILR turned on ? (date)    >>
                          DB+107
  255     00000 1
  256     00000 1   logical ILR'WAS'NEEDED;         << Was ILR needed on first DBOPEN ?   >>
                          DB+110
  257     00000 1
```

```
 258     00000 1   double  ILR'ON'TIME;          << When was ILR turned on ? (time)    >>
                           DB+111
 259     00000 1
 260     00000 1           <<-------------------------------------------------------->>
 261     00000 1           << The next ILR variables control run time handling      >>
 262     00000 1           <<-------------------------------------------------------->>
 263     00000 1
 264     00000 1   logical ILR'ENABLED,          << Is the DB enabled for recovery ?   >>
                           DB+113
 265     00000 1           INTRINSIC'STARTED,    << Header and labels written to log   >>
                           DB+114
 266     00000 1           DATABASE'MOD'COMP;    << DB modified since intrinsic start  >>
                           DB+115
 267     00000 1
 268     00000 1   integer ILR'XDSNUM,           << ILCB(*) number for Recovery        >>
                           DB+116
 269     00000 1           ILR'LABELTBLADR,      << Adr of dset label area in ILCB(*)  >>
                           DB+117
 270     00000 1           ILR'ENTRYCOUNT,       << Number of Entries in the ILCB(*)   >>
                           DB+120
 270.4   00000 1           ILR'MAXILCBBUFFS,       << Max.  number of buffers.  >>
                           DB+121
 271     00000 1           ILR'ENTRYLENGTH,      << Length of an Entry in the ILCB(*)  >>
                           DB+122
 272     00000 1           ILR'CURBLKENTRYS,     << Cur number of entries in Block Tab >>
                           DB+123
 273     00000 1           ILR'MAXBLKENTRYS,     << Max number of entries in Block Tab >>
                           DB+124
 274     00000 1           ILR'DBCBENTRYLEN,     << Length of move from DBCB to ILCB   >>
                           DB+125
 275     00000 1           ILR'NEXTBLKADR;       << Address of next block in ILCB(*)   >>
                           DB+126
 276     00000 1
 277     00000 1
 278     00000 1   LOGICAL ARRAY ENABLED(0:7)=DB; << DATA SET FLAGS - LABEL EVER READ? >>
                           DB+127
 279     00000 1   LOGICAL DBCB'DISABLED;        << DISABLES ACCESS; SET IN DBABORT >>
                           DB+137
 280     00000 1   INTEGER FIXEDBUFFERCOUNT;     << SET BY DBCONTROL; FIXES BUFFERCOUNT >>
                           DB+140
 281     00000 1   LOGICAL OUTPUTDEFERRED;       << QUIESCE I/O FLAG >>
                           DB+141
 282     00000 1   INTEGER ARRAY USERCOUNT(-1:8)=DB; << # ACCESSORS IN EACH MODE >>
                           DB+143
 283     00000 1   LOGICAL DIRTYLABELS;          << FOR QUICK AND DIRTY DBQUIESCE >>
                           DB+154
 284     00000 1   BYTE ARRAY ITEMMAP(0:31)=DB; << 32-BYTE INDEX INTO ITEM TABLE >>
                           DB+155
 285     00000 1   BYTE ARRAY DSETMAP(0:31)=DB; << 32-BYTE INDEX INTO SET TABLE >>
                           DB+175
 286     00000 1
 287     00000 1
 288     00000 1   << LOCKING SYSTEM GLOBAL LOCATIONS >>
 289     00000 1
 290     00000 1   ARRAY DBCBWAIT(0:3)=DB;           << DBCB wait field.  It was a  >>
                           DB+215
 291     00000 1                                     << 2-word wait field.  Since   >>
```



```
 292     00000 1                                     << MPEV, it is 4-word.  For    >>
 293     00000 1                                     << more info. see lock area.   >>
 294     00000 1   INTEGER LOCKAREASIZE;  << WORDS IN TOTAL LOCK BLOCK>>
                           DB+221
 295     00000 1   INTEGER FREESIZE;      << SIZE OF MANAGED FREE SPACE>>
                           DB+222
 296     00000 1   INTEGER MAPSIZE;       << WORDS OF BITMAP >>
                           DB+223
 297     00000 1   INTEGER POINTER FREEAREA;      << POINTS TO START OF MANAGED SPACE>>
                           DB+224
 298     00000 1   INTEGER POINTER MAP;           << POINTS TO START OF BITMAP>>
                           DB+225
 299     00000 1   INTEGER POINTER FIRST'TRY;     << POINTS TO 1ST WORD WITH FREE SPACE>>
                           DB+226
 300     00000 1   INTEGER POINTER BASEENTRY;  <<POINTS AT BASE ENTRY>>
                           DB+227
 301     00000 1   INTEGER POINTER HASHBASE;
                           DB+230
 302     00000 1   INTEGER POINTER WAITMAP;    <<POINTS AT WAITING PROCESS MAP>>
                           DB+231
 303     00000 1
 304     00000 1
 305     00000 1   << SCRATCH POINTERS, COUNTS >>
 306     00000 1
 307     00000 1   INTEGER LOGMODE; << TEMP USED IN LOGGING CALLS >>
                           DB+232
 308     00000 1   INTEGER LOGSTATUS; << TEMP USED IN LOGGING CALLS >>
                           DB+233
 309     00000 1   INTEGER LOGRECLEN;  << LOG RECORD LENGTH (TEMP) >>
                           DB+234
 310     00000 1   INTEGER POINTER LOGBUF; << POINTER TO LOG BUFFER AREA (TEMP) >>
                           DB+235
 311     00000 1   LOGICAL ABORT;                << FLAGS CRITICAL I/O SEQUENCES >>
                           DB+236
 312     00000 1
 313     00000 1   INTEGER ITEMINX;              << "CURRENT" DATA ITEM NUMBER >>
                           DB+237
 314     00000 1   INTEGER POINTER ITEM;         << WD PTR TO ITEM TABLE ENTRY >>
                           DB+240
 315     00000 1
 316     00000 1   INTEGER DSETINX;              << "CURRENT" DATA SET NUMBER >>
                           DB+241
 317     00000 1   INTEGER POINTER DSET;         << WD PTR TO SET TABLE ENTRY >>
                           DB+242
 318     00000 1   LOGICAL MAINT;                << "CURRENT" DATA SET WRITABLE FLAG >>
                           DB+243
 319     00000 1   INTEGER POINTER DSLABEL;      << WD PTR TO DATA SET LABEL DATA >>
                           DB+244
 320     00000 1   INTEGER POINTER DSCB;         << WD PTR TO DSET CONTROL BLOCK >>
                           DB+245
 321     00000 1   INTEGER POINTER ULDSCB;       << WD PTR TO USER LOCAL DSCB >>
                           DB+246
 322     00000 1
 323     00000 1   INTEGER FIELDCT;              << # OF FIELDS, "CURRENT" SET >>
                           DB+247
 324     00000 1   INTEGER FIELDINX;             << "CURRENT" FIELD NUMBER >>
                           DB+250
```

```
 325     00000 1   BYTE POINTER FIELDPTR;        << BYTE PTR TO FIELD # TABLE >>
                       DB+251
 326     00000 1
 327     00000 1   INTEGER PATHINX;              << "CURRENT" PATH NUMBER >>
                       DB+252
 328     00000 1   BYTE POINTER PATHPTR;         << BYTE PTR TO PATH TABLE ENTRY >>
                       DB+253
 329     00000 1
 330     00000 1   INTEGER POINTER DISPL;        << WD PTR TO DISPLACEMENT ARRAY >>
                       DB+254
 331     00000 1
 332     00000 1   BYTE POINTER INUMPTR;         << BYTE PTR TO ITEM NUMBER ARRAY >>
                       DB+255
 333     00000 1
 334     00000 1   INTEGER POINTER BUFFERENTRY;
                       DB+256
 335     00000 1      << POINTER TO CURRENT TABLE ENTRY IN BUFFER AREA >>
 336     00000 1   INTEGER POINTER ILR'CURRENTENTRY;
                       DB+257
 337     00000 1                 << POINTER TO CURRENT ENTRY OF ILR'BLOCK'TABLE >>
 338     00000 1
 339     00000 1   INTEGER POINTER TARGET,ARGUM; << WD PTRS USED IN COMPARES >>
                       DB+260
                       DB+261
 340     00000 1   INTEGER ARGLGTH;              << LENGTH OF ELEMENT AT "ARGUM" >>
                       DB+262
 341     00000 1
 342     00000 1   INTEGER POINTER TEMPDSET;
                       DB+263
 343     00000 1   INTEGER POINTER TEMPDSLABEL;
                       DB+264
 344     00000 1   INTEGER POINTER TEMPDSCB;
                       DB+265
 345     00000 1   INTEGER POINTER TEMPULDSCB;
                       DB+266
 346     00000 1     << POINTERS TO TABLES FOR DATA SET OTHER THAN "CURRENT" ONE >>
 347     00000 1
 348     00000 1   INTEGER POINTER TEMPDISPL;
                       DB+267
 349     00000 1
 350     00000 1   LOGICAL LOCKCOVERS;           << FOR RETURN FROM VERIFYLOCK >>
                       DB+270
 351     00000 1                               << LOCKCOVERS can be replaced with >>
 352     00000 1                               << a local vairalbe.                >>
 353     00000 1
 354     00000 1   INTEGER SCR1;                 << INTEGER TEMPORARIES      >>
                       DB+271
 355     00000 1   INTEGER SCR2;                 <<  USABLE BY ANY           >>
                       DB+272
 356     00000 1   INTEGER SCR3;                 <<   PROCEDURE OR           >>
                       DB+273
 357     00000 1   INTEGER SCR4;                 <<    SUBROUTINE            >>
                       DB+274
 358     00000 1
 359     00000 1   DOUBLE DUBSCRATCH;            << DOUBLE TEMPORARY >>
                       DB+275
 360     00000 1
```



```
 361     00000 1
 362     00000 1
 363     00000 1   << USER FIXED GLOBALS.  THESE ENTITIES, ESTABLISHED AT DBOPEN      >>
 364     00000 1   << TIME, REMAIN FIXED UNTIL THE DATA BASE IS CLOSED.  THEY CAN,    >>
 365     00000 1   << HOWEVER, DIFFER FROM ONE USER TO ANOTHER.                       >>
 366     00000 1
 367     00000 1   INTEGER ARRAY USERFIXEDGLOBALS(0:0)=DB;
                       DB+277
 368     00000 1                 << DUMMY FOR IDENTIFICATION AND INITIALIZATION >>
 369     00000 1   INTEGER CURRENT'USERID;       << WHOSE USER-LOCAL DATA IS THIS? >>
                       DB+300
 370     00000 1   INTEGER CURRENT'OPENCOUNT;
                       DB+301
 371     00000 1   INTEGER DBOPTIONS;            << AOPTIONS FOR DATA SETS >>
                       DB+302
 372     00000 1   INTEGER ARRAY LOGINDEX(0:1)=DB; << LOG INDEX >>
                       OB+303
 373     00000 1   LOGICAL LOGDBMOD;                        << USER IS LOGGING >>
                       DB+305
 374     00000 1   LOGICAL UPDATEOK;             << DBUPDATES OK IN THIS MODE >>
                       DB+306
 375     00000 1   LOGICAL MODIFYOK;             << PUTS-DELS OK IN THIS MODE >>
                       DB+307
 376     00000 1   INTEGER AMODE;                << USER ACCESS (DBOPEN) MODE >>
                       DB+310
 377     00000 1   INTEGER ILR'LOGFILENUM;       << FILE # OF THE LOG FILE >>
                       DB+311
 378     00000 1   INTEGER ROOTNUM;              << FILE # OF OPEN ROOT FILE >>
                       DB+312
 379     00000 1
 380     00000 1
 381     00000 1   INTEGER END'DBCB'GLOBAL;
                       DB+313
 382     00000 1      << THE ADDRESS OF THIS DUMMY VARIABLE IS THE LENGTH >>
 383     00000 1      <<  OF THE DBCB GLOBAL AREA.                         >>
 384     00000 1
 385     00000 1
 386     00000 1   << THE FOLLOWING DECLARATIONS PROVIDE ALTERNATE SYMBOLIC ACCESS TO  >>
 387     00000 1   <<    SOME OF THE ELEMENTS OF THE DBCB GLOBAL AREA DEFINED ABOVE.    >>
 388     00000 1
 389     00000 1   ARRAY              DBCB(*) = DB+0;     << FOR GENERAL DBCB ACCESS >>
                       DB+000
 390     00000 1   DOUBLE POINTER     PHOLD=SCR4;         << SPECIAL FOR CHASESYN >>
                       DB+274
 391     00000 1   INTEGER POINTER P=SCR3;       << SPECIAL FOR FIND'ACCESSOR'ENTRY >>
                       DB+273
 392     00000 1   INTEGER POINTER Q=SCR4;       << SPECIAL FOR FIND'ACCESSOR'ENTRY >>
                       DB+274
 393     00000 1   BYTE ARRAY TAG'B(*) = TAG;
                       DB+000
 394     00000 1   INTEGER TARGETINT=TARGET;     << SPECIAL FOR FOR LOOPS >>
                       DB+260
 395     00000 1   INTEGER BUFFERENTINT=BUFFERENTRY;  << SAME HERE >>
                       DB+256
 396     00000 1
 397     00000 1   $IF X0=ON
 398     00000 1   ARRAY DUMMY(*)=DBCB;
```

```
399      00000 1   $IF
400      00000 1
401      00000 1   DOUBLE POINTER DUBTEMPDSLABEL = TEMPDSLABEL;
                          DB+264
402      00000 1   DOUBLE POINTER DUBTEMPDSCB = TEMPDSCB;
                          DB+265
403      00000 1   DOUBLE POINTER DUBTEMPULDSCB = TEMPULDSCB;
                          DB+266
404      00000 1   BYTE ARRAY REMDBNAMEBYTE(*) = REMDBNAME;
                          DB+024
405      00000 1
406      00000 1
407      00000 1   << THE FOLLOWING DECLARATIONS ARE ALTERNATE DECLARATIONS OF THE >>
408      00000 1   <<  DBCB GLOBAL AREA USED WHEN ACCESSING A REMOTE DATA BASE.    >>
409      00000 1
410      00000 1   INTEGER POINTER LASTLISTSIZE = ENABLED;
                          DB+127
411      00000 1      << POINTS TO ARRAY OF DATA SET LAST LIST SIZES >>
412      00000 1   INTEGER DSDEVNAM = ENABLED+1; << USES 5 WORDS >>
                          DB+130
413      00000 1   BYTE ARRAY DSLNUM(*) = DSDEVNAM+1;    << DS DEVICE >>
                          DB+131
414      00000 1   ARRAY DSLNUMW(*) = DSLNUM;
                          DB+131
415      00000 1   LOGICAL IGOTDBFLAG = ENABLED+6; << IS REMOTE DATA BASE LOCKED? >>
                          DB+135
416      00000 1
417      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  >>
418      00000 1   << END OF DATA BASE GLOBAL AREA.  >>
419      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  >>
```



```
421      00000 1   <<                  ****************************
422      00000 1                       *    ITEM SECURITY TABLE     *
423      00000 1                       ****************************
424      00000 1                       *    SET SECURITY TABLE      *
425      00000 1                       ****************************                    >>
426      00000 1
427      00000 1
428      00000 1
429      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
430      00000 1   << THESE DECLARATIONS DEFINE THE FORMAT OF, AND CONTROL ACCESS TO
431      00000 1   THE ITEM AND SET SECURITY TABLES.  THESE TABLES IMMEDIATELY FOLLOW
432      00000 1   THE GLOBAL AREA, AND CONSIST OF 1 WORD FOR EACH DATA ITEM FOLLOWED
433      00000 1   BY ONE WORD FOR EACH DATA SET.  THE ZEROETH ENTRIES (WHICH DON'T
434      00000 1   ACTUALLY EXIST) OF THE TWO ARRAYS ARE POINTED TO BY ITEMSECTBL
435      00000 1   AND DSETSECTBL RESPECTIVELY.  THE FOLLOWING SYMBOLS DEPEND UPON
436      00000 1   PRIOR SETTING OF ITEMINX OR DSETINX. >>
437      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
438      00000 1
439      00000 1   DEFINE READABLE = 2:1#;
440      00000 1   DEFINE WRITABLE = 10:1#;
441      00000 1   DEFINE READBITS = 2:2#;
442      00000 1
443      00000 1   DEFINE
444      00000 1
445      00000 1   ITEMREADBYTE    = ITEMSECTBL(ITEMINX).(0:8)#, << ITEM READ BYTE >>
446      00000 1   ITEMREADABLE    = ITEMSECTBL(ITEMINX).(2:1)#, << READABLE IN A SET >>
447      00000 1   ITEMREADMSKBIT  = ITEMSECTBL(ITEMINX).(3:1)#, << IN READ CLASS >>
448      00000 1   ITEMREADBITS    = ITEMSECTBL(ITEMINX).(2:2)#, << BOTH READ BITS >>
449      00000 1   ITEMREADLEVEL   = ITEMSECTBL(ITEMINX).(4:4)#, << ITEM READ LEVEL >>
450      00000 1   ITEMWRITEBYTE   = ITEMSECTBL(ITEMINX).(8:8)#, << ITEM WRITE BYTE >>
451      00000 1   ITEMWRITABLE    = ITEMSECTBL(ITEMINX).(10:1)#,<< WRITABLE IN A SET >>
452      00000 1   ITEMWRITMSKBIT  = ITEMSECTBL(ITEMINX).(11:1)#, << IN WRITE CLASS >>
453      00000 1   ITEMWRITEBITS   = ITEMSECTBL(ITEMINX).(10:2)#, << BOTH WRITE BITS >>
454      00000 1   ITEMWRITELEVEL  = ITEMSECTBL(ITEMINX).(12:4)#, << ITEM WRITE LEVEL >>
455      00000 1
456      00000 1   DSETREADBYTE    = DSETSECTBL(DSETINX).(0:8)#, << SET READ BYTE >>
457      00000 1   DSETREADABLE    = DSETSECTBL(DSETINX).(2:1)#, << HAS READABLE ITEMS >>
458      00000 1   DSETREADMSKBIT  = DSETSECTBL(DSETINX).(3:1)#, << IN READ CLASS >>
459      00000 1   DSETREADBITS    = DSETSECTBL(DSETINX).(2:2)#, << BOTH READ BITS >>
460      00000 1   DSETREADLEVEL   = DSETSECTBL(DSETINX).(4:4)#, << SET READ LEVEL >>
461      00000 1   DSETWRITEBYTE   = DSETSECTBL(DSETINX).(8:8)#, << SET WRITE BYTE >>
462      00000 1   DSETWRITABLE    = DSETSECTBL(DSETINX).(10:1)#, << IS WRITABLE >>
463      00000 1   DSETWRITMSKBIT  = DSETSECTBL(DSETINX).(11:1)#, << IN WRITE CLASS >>
464      00000 1   DSETWRITELEVEL  = DSETSECTBL(DSETINX).(12:4)#; << SET WRITE LEVEL >>
465      00000 1
466      00000 1   << THE READ AND WRITE BYTES OF THE ITEM AND SET SECURITY TABLES       >>
467      00000 1   <<     (ABOVE) HAVE THE FOLLOWING BREAKDOWN:                          >>
468      00000 1   <<          0TH BIT:       UNUSED                                     >>
469      00000 1   <<          1ST BIT:       UNUSED                                     >>
470      00000 1   <<          2ND BIT:       SET BY DBOPEN;  1 MEANS THIS ELEMENT HAS   >>
471      00000 1   <<                         AN ACTUAL OCCURENCE WHICH IS ACCESSIBLE.   >>
472      00000 1   <<                         (FOR EXAMPLE, AN ITEM IS READABLE AND IS IN  >>
473      00000 1   <<                         SOME SET WHICH IS READABLE.                >>
474      00000 1   <<          3RD BIT:       SET BY DBOPEN;  1 MEANS THE USER'S ACCESS  >>
475      00000 1   <<                         CLASS IS INCLUDED IN THIS ELEMENT'S MASK, OR >>
476      00000 1   <<                         THAT THE USER HAS ACCESS TO EVERYTHING.    >>
477      00000 1   <<          4TH-7TH BITS:  SET BY DBSCHEMA;  FOR OLD SCHEMATA, THE    >>
```

```
478     00000 1   <<                        ELEMENT'S LEVEL;  FOR NEW SCHEMATA, ZERO.    >>
479     00000 1
480     00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
481     00000 1   << END OF ITEM AND SET SECURITY TABLES >>
482     00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
```



```
484     00000 1   <<                  ****************************
485     00000 1                       *                          *
486     00000 1                       *                          *
487     00000 1                       *        ITEM TABLE        *
488     00000 1                       *                          *
489     00000 1                       *                          *
490     00000 1                       ****************************
491     00000 1                       *                          *
492     00000 1                       *                          *
493     00000 1                       *         SET TABLE        *
494     00000 1                       *                          *
495     00000 1                       *                          *
496     00000 1                       ****************************                    >>
497     00000 1
498     00000 1
499     00000 1
500     00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
501     00000 1   << THESE DECLARATIONS DEFINE THE FORMAT OF, AND CONTROL ACCESS TO,    >>
502     00000 1   <<   THE FIXED-LENGTH ENTRIES IN THE ITEM AND SET TABLES.  THE 0TH    >>
503     00000 1   <<   ENTRIES OF THESE TABLES ARE POINTED TO BY ITEMTBL AND DSETTBL,   >>
504     00000 1   <<   RESPECTIVELY.  FOR THE DEFINES GIVEN HERE TO BE USED CORRECTLY,  >>
505     00000 1   <<   INTEGER POINTER ITEM OR INTEGER POINTER DSET MUST BE SET TO A    >>
506     00000 1   <<   LEGITIMATE ENTRY IN ITS RESPECTIVE TABLE.  THIS CAN BE          >>
507     00000 1   <<   ACCOMPLISHED BY ESTABLISHING AN ITEM NUMBER IN ITEMINX OR SET   >>
508     00000 1   <<   NUMBER IN DSETINX, AND THEN USING THE DEFINE FIXITEM OR FIXDSET. >>
509     00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
510     00000 1
511     00000 1   << FIRST, OFFSETS INTO ITEM AND DSET >>
512     00000 1
513     00000 1   EQUATE    OF'TABLE'OFFSETS=0, << DUMMY TO IDENTIFY THIS STUFF >>
514     00000 1
515     00000 1             OFNM       = 0,      << ITEM OR SET NAME >>
516     00000 1             OFSY       = 8,      << SYNONYM NUMBER, ITEM OR SET TYPE >>
517     00000 1             OFUU       = 9,      << FOR ITEMS, UNUSED >>
518     00000 1             OFUL       = 9,      << USER LOCAL DSCB POINTER >>
519     00000 1             OFSC       = 10,     << SUBITEM COUNT AND LENGTH >>
520     00000 1             OFDS       = 10,     << DATA SET CONTROL BLOCK POINTER >>
521     00000 1             ENTRYSIZE  = 11;     << OVERALL TABLE ENTRY LENGTH >>
522     00000 1
523     00000 1   << SYMBOLIC NAMES FOR ITEM TABLE ELEMENTS.  INTEGER POINTER "ITEM" >>
524     00000 1   <<   *MUST* BE SET FOR VALID USE OF THESE SYMBOLS.                  >>
525     00000 1
526     00000 1   INTEGER POINTER INTITEM = ITEM;    << INTERMEDIATE FOR DEFINES >>
                        DB+240
527     00000 1
528     00000 1   DEFINE    ITEMNAME       = INTITEM(OFNM)#,       << 16 BYTE NAME >>
529     00000 1             ITEMSYNNUM     = INTITEM(OFSY).(0:8)#, << # OF "SYNONYM" >>
530     00000 1             ITEMTYPE       = INTITEM(OFSY).(8:8)#, << I J K R X U Z P >>
531     00000 1             ITEMUNUSED     = INTITEM(OFUU)#,       << UNUSED >>
532     00000 1             ITEMSUBCT      = INTITEM(OFSC).(0:8)#, << SUBITEM COUNT >>
533     00000 1             ITEMSUBLGTH    = INTITEM(OFSC).(8:8)#; << SUBITEM LENGTH >>
534     00000 1
535     00000 1   << SYMBOLIC NAMES FOR DSET TABLE ELEMENTS.  INTEGER POINTER "DSET" >>
536     00000 1   <<   *MUST* BE SET FOR VALID USE OF THESE SYMBOLS.                  >>
537     00000 1
538     00000 1   INTEGER POINTER INTDSET = DSET;    << INTERMEDIATE FOR DEFINES >>
                        DB+242
```

```
539      00000 1
540      00000 1   DEFINE    DSETNAME       = INTDSET(OFNM)#,        << 16 BYTE NAME >>
541      00000 1             DSETSYNNUM     = INTDSET(OFSY).(0:8)#,  << # OF "SYNONYM" >>
542      00000 1             DSETTYPE       = INTDSET(OFSY).(8:8)#,  << SET TYPE: D M A >>
543      00000 1             DSETULDSCB     = INTDSET(OFUL)#,        << ULDSCB POINTER >>
544      00000 1             DSETDSCB       = INTDSET(OFDS)#;        << DSCB POINTER >>
545      00000 1
546      00000 1   <<  THE ITEM AND SET TABLES HAVE A SLIGHTLY DIFFERENT                  >>
547      00000 1   <<  FORMAT WHEN DEALING WITH A REMOTE DATA BASE.  THE                  >>
548      00000 1   <<  DEFINITIONS GIVEN ABOVE TAKE ON THE FOLLOWING                     >>
549      00000 1   <<  MEANINGS  :                                                      >>
550      00000 1   <<                                                                   >>
551      00000 1   <<       OFNM       =  0        NAME                                 >>
552      00000 1   <<       OFSY       =  8        SYNONYM NUMBER, UNUSED BYTE          >>
553      00000 1   <<       OFUL       =  9        SIZE OF DATA SET KEY                 >>
554      00000 1   <<                                (DETAIL SET = 0)                   >>
555      00000 1   <<       OFSC=OFDS  = 10        SIZE OF ENTRY (WORDS)                >>
556      00000 1   <<                                                                   >>
557      00000 1
558      00000 1   <<  THE FOLLOWING DEFINITIONS ARE ALSO NECESSARY       >>
559      00000 1
560      00000 1   DEFINE  ITEMREMLGTH   = INTITEM(OFSC)#,   << ITEM SIZE >>
561      00000 1           DSETREMLGTH    = INTDSET(OFDS)#,   << SET SIZE  >>
562      00000 1           DSETREMKEYSIZE= INTDSET(OFUL)#;   << KEY SIZE  >>
563      00000 1
564      00000 1   <<  *  *  *  *  *  *  *  *  *  *  >>
565      00000 1   << END OF ITEM AND SET TABLES. >>
566      00000 1   <<  *  *  *  *  *  *  *  *  *  *  >>
```



```
568      00000 1   <<                  ****************************
569      00000 1                       *                          *
570      00000 1                       *   DATA SET LABEL TABLE   *
571      00000 1                       *                          *
572      00000 1                       ****************************
573      00000 1                       *                          *
574      00000 1                       *        DATA SET          *
575      00000 1                       *     CONTROL BLOCKS       *
576      00000 1                       *       (DSCB'S)           *
577      00000 1                       *                          *
578      00000 1                       ****************************
579      00000 1                       *                          *
580      00000 1                       *       USER LOCAL         *
581      00000 1                       *        DATA SET          *
582      00000 1                       *     CONTROL BLOCKS       *
583      00000 1                       *       (ULDSCB'S)         *
584      00000 1                       *                          *
585      00000 1                       ****************************                >>
586      00000 1
587      00000 1
588      00000 1
589      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
590      00000 1   <<   FOLLOWING THE ITEM AND SET TABLES ARE 3 TABLES WHICH RELATE TO
591      00000 1   DATA SETS:
592      00000 1              THE DATA SET LABEL TABLE (DSLABELTBL)
593      00000 1              THE DATA SET CONTROL BLOCK (DSCB) TABLE
594      00000 1            AND THE USER LOCAL DATA SET CONTROL BLOCK (ULDSCB) TABLE
595      00000 1
596      00000 1   THE DSLABELTBL CONTAINS THE DYNAMIC, GLOBAL INFORMATION ABOUT
597      00000 1   DATA SETS WHICH MUST BE KEPT WITH THE DATA BASE PERMANENTLY.  THIS
598      00000 1   INCLUDES A DATA SET'S END-OF-FILE (ACTUALLY CHANGES ONLY FOR DETAILS),
599      00000 1   FREE RECORD COUNT, AND FREE RECORD CHAIN HEAD (PERTINENT ONLY FOR
600      00000 1   DETAILS).  THIS AMOUNTS TO THREE (3) DOUBLE-WORD VALUES PER DATA
601      00000 1   SET, AND IS PRESENTLY STORED IN THE DATA SET'S USER LABEL.
602      00000 1
603      00000 1   THE DSCB FOR EACH DATA SET HAS A LENGTH WHICH DEPENDS UPON THE
604      00000 1   NUMBER OF FIELDS AND PATHS IN THE SET.  IT CONTAINS FIXED INFOR-
605      00000 1   MATION ABOUT THE DATA SET DERIVED FROM THE SCHEMA, AND THUS IS
606      00000 1   THE SAME FOR EACH USER.
607      00000 1
608      00000 1   THE ULDSCB FOR EACH DATA SET HAS A LENGTH WHICH VARIES WITH THE
609      00000 1   NUMBER OF FIELDS IN THE SET.  IT CONTAINS DYNAMIC INFORMATION
610      00000 1   RELATING TO THE USER'S CURRENT ACCESS TO THE DATA SET (SUCH AS
611      00000 1   CURRENT RECORD POINTER), SO IT IS GENERALLY DIFFERENT FOR EACH
612      00000 1   USER.
613      00000 1
614      00000 1   THE FOLLOWING DECLARATIONS DEFINE THE LAYOUTS OF THESE TABLES. >>
615      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
```

```
617     00000 1   <<                  ****************************
618     00000 1                       *                          *
619     00000 1                       *   DATA SET LABEL TABLE   *
620     00000 1                       *                          *
621     00000 1                       ****************************                    >>
622     00000 1
623     00000 1
624     00000 1
625     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
626     00000 1   <<   THESE DECLARATIONS DEFINE THE FORMAT OF, AND CONTROL ACCESS TO,
627     00000 1   THE ENTRIES OF THE DATA SET LABEL TABLE.  FOR EACH DATA SET, THE
628     00000 1   ENTRY CONSISTS OF 3 DOUBLE-WORD ENTITIES.  THE 0TH ENTRY OF THE TABLE
629     00000 1   (WHICH DOESN'T REALLY EXIST) IS POINTED TO BY THE DBCB GLOBAL POINTER
630     00000 1   "DSLABELTBL".  FOR THE FOLLOWING IDENTIFIERS TO BE USED, INTEGER
631     00000 1   POINTER "DSLABEL" *MUST* BE SET TO THE 6-WORD DATA SET ENTRY OF
632     00000 1   INTEREST. >>
633     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
634     00000 1
635     00000 1   << FIRST, DOUBLE WORD OFFSETS INTO A DSLABEL >>
636     00000 1
637     00000 1   EQUATE OL'DSLABEL'OFFSETS = 0, << DUMMY TO IDENTIFY THIS JUNK >>
638     00000 1
639     00000 1         OLEN    = 0,     << END-OF-FILE NAME >>
640     00000 1         OLFC    = 1,     << FREE COUNT >>
641     00000 1         OLFH    = 2;     << FREE HEAD >>
642     00000 1
643     00000 1       <<  ************************************
644     00000 1   INTEGER POINTER DSLABEL'ELEMENT = DSLABELTBL;
645     00000 1      << DEFINE USED TO ACCESS A 6-WORD ENTRY AS A UNIT >>
646     00000 1
647     00000 1   << NOW, SYMBOLIC NAMES FOR DSLABEL ENTRY ELEMENTS.  INTEGER POINTER >>
648     00000 1   <<  "DSLABEL"  *MUST*  BE SET FOR VALID USE OF THESE SYMBOLS.       >>
649     00000 1
650     00000 1   DOUBLE POINTER DUBDSLABEL = DSLABEL;
                       DB+244
651     00000 1      << THIS IDENTIFIER FORMS THE BASIS FOR DSLABEL-RELATIVE >>
652     00000 1      <<          DECARATIONS.                                 >>
653     00000 1
654     00000 1   DEFINE
655     00000 1
656     00000 1   DSEOFNAME       = DUBDSLABEL(OLEN)#, << END-OF-FILE NAME >>
657     00000 1   DSFREECT        = DUBDSLABEL(OLFC)#; << FREE RECORD COUNT >>
658     00000 1   DSFREEHD        = DUBDSLABEL(OLFH)#; << FREE RECORD CHAIN HEAD >>
659     00000 1
660     00000 1
661     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
662     00000 1   << END OF DATA SET LABEL TABLE >>
663     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
```



```
665     00000 1   <<                  ****************************
666     00000 1                       *                          *
667     00000 1                       *        DATA SET          *
668     00000 1                       *     CONTROL BLOCKS       *
669     00000 1                       *       (DSCB'S)           *
670     00000 1                       *                          *
671     00000 1                       ****************************                    >>
672     00000 1
673     00000 1
674     00000 1
675     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
676     00000 1   << THESE DECLARATIONS DEFINE THE FORMAT OF, AND CONTROL ACCESS TO,    >>
677     00000 1   <<   THE HEADER (GLOBAL) PORTION OF EACH DATA SET CONTROL BLOCK       >>
678     00000 1   <<   (DSCB).  A POINTER TO A DATA SET'S DSCB IS OBTAINED FROM THE     >>
679     00000 1   <<   FIXED-ENTRY-LENGTH DATA SET TABLE (DSETTBL); INTEGER POINTER     >>
680     00000 1   <<   DSCB MUST BE SET FOR THESE DEFINES TO BE VALID.                  >>
681     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
682     00000 1
683     00000 1   << FIRST, OFFSETS (WORD OR DOUBLE WORD, AS REQUIRED) INTO DSCB >>
684     00000 1
685     00000 1   EQUATE    OS'DSCB'OFFSETS =0, << DUMMY TO IDENTIFY THIS JUNK >>
686     00000 1
687     00000 1             OSCP       = 0,      << CAPACITY >>
688     00000 1             OSBL       = 2,      << BLOCK LENGTH >>
689     00000 1             OSML       = 3,      << MEDIA RECORD LENGTH >>
690     00000 1             OSEL       = 4,      << ENTRY LENGTH >>
691     00000 1             OSBF       = 5,      << BLOCKING FACTOR, FIELD COUNT >>
692     00000 1             OSPC       = 6,      << PATH COUNT, PRIMARY PATH OR KEY FIELD>>
693     00000 1             OSPP       = 7,      << OFFSET TO PATH TABLE >>
694     00000 1             OSSN       = 4,      << "SAVENAME" >>
695     00000 1             DSCBHDRSIZE = 10;    << OVERALL DSCB HEADER LENGTH >>
696     00000 1
697     00000 1   << SYMBOLIC NAMES FOR DSCB HEADER ELEMENTS.  INTEGER POINTER "DSCB" >>
698     00000 1   <<   *MUST* BE SET FOR VALID USE OF THESE SYMBOLS.                   >>
699     00000 1
700     00000 1   INTEGER POINTER INTDSCB = DSCB;     << THESE FORM BASIS FOR ALL      >>
                       DB+245
701     00000 1   DOUBLE POINTER  DUBDSCB = DSCB;     <<   DSCB-RELATIVE DECLARATIONS. >>
                       DB+245
702     00000 1
703     00000 1   DEFINE     DSCAP          = DUBDSCB(OSCP)#,        << CAPACITY >>
704     00000 1             DSBLOCKLGTH    = INTDSCB(OSBL)#,        << BLOCK LENGTH >>
705     00000 1             DSMEDIALGTH    = INTDSCB(OSML)#,        << MEDIA RECORD LGTH>>
706     00000 1             DSENTRYLGTH    = INTDSCB(OSEL)#,        << ENTRY LENGTH >>
707     00000 1             DSBLOCKFAC     = INTDSCB(OSBF).(0:8)#,  << BLOCKING FACTOR >>
708     00000 1             DSFIELDCT      = INTDSCB(OSBF).(8:8)#,  << # ITEMS/RECORD >>
709     00000 1             DSPATHCT       = INTDSCB(OSPC).(0:8)#,  << # PATHS FOR SET >>
710     00000 1             DSKEYTYPE      = INTDSCB(OSPC).(8:1)#,  << KEY NO HASH/HASH >>
711     00000 1             DSPRIMKEY      = INTDSCB(OSPC).(9:7)#,  << PRIM PATH OR KEY >>
712     00000 1             DSPATHPTR      = INTDSCB(OSPP)#,        << OFFSET TO PATH TB>>
713     00000 1             DSSAVENAME     = DUBDSCB(OSSN)#;        << SCRATCH AREA >>
714     00000 1
715     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
716     00000 1   << END OF DATA SET CONTROL BLOCK HEADER.>>
717     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
```

```
719     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
720     00000 1   << WITHIN EACH DSCB, THE GLOBAL AREA (HEADER PORTION) IS FOLLOWED                >>
721     00000 1   <<   BY THE RECORD DEFINITION (RECORD XFORM) TABLES.  THESE CONSIST              >>
722     00000 1   <<   OF TWO ARRAYS, AS FOLLOWS:                                                 >>
723     00000 1   <<                                                                             >>
724     00000 1   <<        DISPLACEMENT ARRAY: THE MEDIA RECORD OFFSETS OF EACH FIELD           >>
725     00000 1   <<        ITEM NUMBER ARRAY: THE ITEM NUMBER OF EACH FIELD IN THE SET          >>
726     00000 1   <<                                                                             >>
727     00000 1   <<   THE DISPLACEMENT ARRAY IS A WORD ARRAY, AND IS TERMINATED BY THE          >>
728     00000 1   <<   WORD OFFSET OF THE END OF THE MEDIA RECORD (IN OTHER WORDS, THE           >>
729     00000 1   <<   MEDIA RECORD LENGTH).  THE ITEM NUMBER ARRAY IS A BYTE ARRAY,            >>
730     00000 1   <<   TERMINATED BY ONE OR TWO BYTES OF ZERO.  THE ARRAYS ARE                   >>
731     00000 1   <<   GENERALLY ACCESSED BY SETTING POINTERS DISPL AND INUMPTR,               >>
732     00000 1   <<   RESPECTIVELY.  (SUBROUTINE FIXRECPTRS ACCOMPLISHES THIS, BASED          >>
733     00000 1   <<   ON THE SETTING OF DSCB.                                                  >>
734     00000 1   <<                                                                             >>
735     00000 1   << THE FINAL ELEMENT OF EACH DSCB IS THE PATH TABLE.  THIS CONTAINS           >>
736     00000 1   <<   ONE 4-BYTE ENTRY FOR EACH PATH DEFINED FOR THE DATA SET.  THE           >>
737     00000 1   <<   CONTENTS ARE:                                                           >>
738     00000 1   <<                                                                             >>
739     00000 1   <<        FOR MASTER DATA SETS:                                                >>
740     00000 1   <<      BYTE 1:  ITEM NUMBER OF SEARCH ITEM IN RELATED DETAIL                  >>
741     00000 1   <<      BYTE 2:  ITEM NUMBER OF SORT ITEM IN RELATED DETAIL (0=NONE)           >>
742     00000 1   <<      BYTE 3:  SET NUMBER OF RELATED DETAIL DATA SET                         >>
743     00000 1   <<      BYTE 4:  PATH NUMBER OF CORRESPONDING PATH IN RELATED DETAIL           >>
744     00000 1   <<        FOR DETAIL DATA SETS:                                                >>
745     00000 1   <<      BYTE 1:  FIELD NUMBER OF SEARCH ITEM                                   >>
746     00000 1   <<      BYTE 2:  FIELD NUMBER OF SORT ITEM (0 IF NONE)                         >>
747     00000 1   <<      BYTE 3:  SET NUMBER OF RELATED MASTER DATA SET                         >>
748     00000 1   <<      BYTE 4:  PATH NUMBER OF CORRESPONDING PATH IN RELATED MASTER            >>
749     00000 1   <<                                                                             >>
750     00000 1   <<   THE PATH TABLE ENTRIES ARE GENERALLY ACCESSED THROUGH USE OF             >>
751     00000 1   <<   PATHINX AND BYTE POINTER PATHPTR.  SEVERAL DEFINES MANIPULATE            >>
752     00000 1   <<   THESE VALUES.                                                            >>
753     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
754     00000 1
755     00000 1
756     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  >>
757     00000 1   << END OF DATA SET CONTROL BLOCK. >>
758     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  >>
```



```
760     00000 1   <<                  ****************************
761     00000 1                       *                          *
762     00000 1                       *        USER LOCAL        *
763     00000 1                       *         DATA SET         *
764     00000 1                       *      CONTROL BLOCKS      *
765     00000 1                       *        (ULDSCB'S)        *
766     00000 1                       *                          *
767     00000 1                       ****************************                          >>
768     00000 1
769     00000 1
770     00000 1
771     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
772     00000 1   <<   THESE DECLARATIONS DEFINE THE FORMAT OF, AND CONTROL ACCESS
773     00000 1   TO, THE HEADER (GLOBAL) PORTION OF EACH USER LOCAL DATA SET CONTROL
774     00000 1   BLOCK (ULDSCB).  A POINTER TO THE DATA SET'S ULDSCB IS OBTAINED
775     00000 1   FROM THE FIXED-ENTRY-LENGTH DATA SET TABLE (DSETTBL);  INTEGER
776     00000 1   POINTER ULDSCB MUST BE SET FOR THE DEFINES BELOW TO BE VALID. >>
777     00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
778     00000 1
779     00000 1   << FIRST, OFFSETS (WORD OR DOUBLE WORD, AS REQUIRED) INTO ULDSCB >>
780     00000 1
781     00000 1   EQUATE OU'ULDSCB'OFFSETS = 0, <<DUMMY TO IDENTIFY THIS JUNK >>
782     00000 1
783     00000 1         OUCD   = 0,    << CONDITION WORD >>
784     00000 1         OUDL   = 1,    << DATA TRANSFER LENGTH >>
785     00000 1         OURN   = 1,    << RECORD NAME >>
786     00000 1         OUCL   = 2,    << CHAIN LENGTH >>
787     00000 1         OUBN   = 3,    << BACKWARD NAME >>
788     00000 1         OUFN   = 4,    << FORWARD NAME >>
789     00000 1         OUST   = 10,   << STATUS BITS >>
790     00000 1         OUPN   = 11,   << PATH NUMBER, OPCODE >>
791     00000 1         OUOM   = 12,   << OPMODE, FILE NUMBER >>
792     00000 1
793     00000 1         ULDSCBHDRSIZE  =  13;  << OVERALL ULDSCB HEADER LENGTH >>
794     00000 1
795     00000 1
796     00000 1   << SYMBOLIC NAMES FOR ULDSCB HEADER ELEMENTS.  INTEGER POINTER >>
797     00000 1   << "ULDSCB"  *MUST*  BE SET FOR VALID USE OF THESE SYMBOLS.    >>
798     00000 1
799     00000 1   INTEGER POINTER INTULDSCB = ULDSCB;
                        DB+246
800     00000 1   DOUBLE POINTER DUBULDSCB = ULDSCB;
                         DB+246
801     00000 1      << ABOVE IDENTIFIERS USED FOR ALL ULDSCB-RELATIVE DECLARATIONS >>
802     00000 1
803     00000 1   DEFINE
804     00000 1
805     00000 1   DSCONWORD       = INTULDSCB(OUCD)#,       << CONDITION WORD >>
806     00000 1   DSDATALGTH      = INTULDSCB(OUDL)#,       << DATA XFER LENGTH >>
807     00000 1   DSRECNAME       = DUBULDSCB(OURN)#,       << CURRENT RECORD NAME >>
808     00000 1   DSCHAINLGTH     = DUBULDSCB(OUCL)#,       << CHAIN LENGTH >>
809     00000 1   DSBWDNAME       = DUBULDSCB(OUBN)#,       << BACKWARD POINTER >>
810     00000 1   DSFWDNAME       = DUBULDSCB(OUFN)#,       << FORWARD POINTER >>
811     00000 1   DSSTATUS        = INTULDSCB(OUST)#,       << STATUS BITS >>
812     00000 1   DSPATHNUM       = INTULDSCB(OUPN).(0:8)#, << CURRENT PATH >>
813     00000 1   DSOPCODE        = INTULDSCB(OUPN).(8:8)#, << LAST PROCEDURE >>
814     00000 1   DSOPMODE        = INTULDSCB(OUOM).(0:8)#, << LAST MODE >>
```

```
 815      00000 1   DSFILENUM       = INTULDSCB(OUOM).(8:8)#; << DSET'S FILE NUMBER >>
 816      00000 1
 817      00000 1   DEFINE    FULLREC        = DSSTATUS.(15:1)#,      << LIST WAS "@;" >>
 818      00000 1             PUTREADY       = DSSTATUS.(14:1)#,      << LIST: OK FOR PUTS>>
 819      00000 1             DIAGNOSTIC     = DSSTATUS.(11:2)#,    << 2 => LIST="?;" >>
 820      00000 1                                                   << 1 => LIST="/;" >>
 821      00000 1                                                   << 0 => otherwise >>
 822      00000 1             FULLPUT        = DSSTATUS.(14:2)#;      << 2 OF THE ABOVE >>
 823      00000 1
 824      00000 1
 825      00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
 826      00000 1   << END OF USER LOCAL DATA SET CONTROL BLOCK HEADER >>
 827      00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
```



```
 829      00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
 830      00000 1   <<   WITHIN EACH ULDSCB, THE GLOBAL AREA (HEADER PORTION) IS
 831      00000 1   FOLLOWED BY THE CURRENT LIST ARRAY FOR THIS DATA SET (FOR THIS
 832      00000 1   USER).  THIS IS A BYTE ARRAY CALLED THE FIELD ARRAY, AND CONSISTS
 833      00000 1   OF THE USER'S CURRENT LIST BY FIELD NUMBER (NOT ITEM NUMBER).  IT
 834      00000 1   IS TERMINATED BY ONE OR TWO BYTES OF ZERO.  THE FIELD ARRAY IS
 835      00000 1   ACCESSED BY SETTING THE GLOBAL DBCB BYTE POINTER "FIELDPTR".
 836      00000 1   SUBROUTINE FIXRECPTRS ACCOMPLISHES THIS, BASED ON THE SETTING
 837      00000 1   OF ULDSCB. >>
 838      00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
 839      00000 1
 840      00000 1
 841      00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
 842      00000 1   << END OF USER LOCAL DATA SET CONTROL BLOCK >>
 843      00000 1   << * * * * * * * * * * * * * * * * * * * * * * * * * * * >>
```

```
 845      00000 1   <<                  ****************************
 846      00000 1                       *   USER DYNAMIC GLOBALS     *
 847      00000 1                       ****************************                    >>
 848      00000 1
 849      00000 1
 850      00000 1
 851      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
 852      00000 1   << IMMEDIATELY FOLLOWING THE USER LOCAL DATA SET CONTROL BLOCKS IS
 853      00000 1   THE ONLY OTHER AREA WHICH CONTAINS DATA WHICH IS BOTH "DYNAMIC"
 854      00000 1   AND "USER LOCAL".  THIS AREA, CALLED THE USER DYNAMIC GLOBAL AREA,
 855      00000 1   IS POINTED TO BY THE DBCB GLOBAL POINTER "USERDYNAMICAREA".  THE
 856      00000 1   DATA IN THIS AREA APPLIES NOT TO A PARTICULAR DATA SET, BUT TO THIS
 857      00000 1   USER'S ACCESS TO THE DATA BASE AS A WHOLE.  (THUS, IT IS CALLED
 858      00000 1   "GLOBAL".)  IT CHANGES FROM CALL TO CALL (THUS "DYNAMIC"), AND
 859      00000 1   GENERALLY DIFFERS FOR EACH USER (THUS "USER").  ALL VARIABLES
 860      00000 1   WITHIN THE USER DYNAMIC GLOBAL AREA ARE ACCESSED VIA THE POINTER
 861      00000 1   USERDYNAMICAREA, OR ONE OF ITS EQUIVALENTS.  THE FOLLOWING
 862      00000 1   DECLARATIONS DEFINE AND CONTROL ACCESS TO THESE ELEMENTS.    >>
 863      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
 864      00000 1
 865      00000 1   << FIRST, IDENTIFIERS FOR USE IN USERDYNAMICAREA-RELATIVE DECLARATIONS>>
 866      00000 1
 867      00000 1   INTEGER POINTER INTUSERDYNAMIC = USERDYNAMICAREA;
                            DB+013
 868      00000 1   LOGICAL POINTER LOGUSERDYNAMIC = USERDYNAMICAREA;
                            DB+013
 869      00000 1   DOUBLE  POINTER DUBUSERDYNAMIC = USERDYNAMICAREA;
                            DB+013
 870      00000 1
 871      00000 1   DEFINE
 872      00000 1
 873      00000 1   LOGTRANSNUM = DUBUSERDYNAMIC(0)#, << TRANSACTION NUMBER COUNT >>
 874      00000 1   TRANSACTING = LOGUSERDYNAMIC(2)#; << ACTIVE TRANSACTION FLAG >>
 875      00000 1
 876      00000 1   EQUATE USERDYNGLOBSIZE = 3; << WORDS IN THIS AREA >>
```



```
 878      00000 1   <<                  **********************
 879      00000 1                       *                      *
 880      00000 1                       *                      *
 881      00000 1                       *  ILR BLOCK TABLE     *
 882      00000 1                       *                      *
 883      00000 1                       *                      *
 884      00000 1                       **********************                        >>
 885      00000 1
 886      00000 1
 887      00000 1   <<   This is a specialized table for use only by the intrinsics >>
 888      00000 1   << which modify the database.  This table allows us to keep     >>
 889      00000 1   << track of the blocks which have copies in the ILCB and    >>
 890      00000 1   << which blocks have been written to the database log file so   >>
 891      00000 1   << that no BLOCK from a data set is ever copied twice.          >>
 892      00000 1
 893      00000 1   <<   This table is pointed to by the global DBCB pointer       >>
 894      00000 1   << called:   "ILR'BLOCK'TABLE"                                  >>
 895      00000 1
 896      00000 1   <<   The table is divided into two (2) physical areas:         >>
 897      00000 1
 898      00000 1   <<     1)    Index area  - equal to the number of data sets     >>
 899      00000 1   <<                         in this database plus the zero'th    >>
 900      00000 1   <<                         which is used for the next available >>
 901      00000 1   <<                         free four (4)-word entry.            >>
 902      00000 1
 903      00000 1   <<     2)    Record area - equal to the maximum number of      >>
 904      00000 1   <<                         blocks that could be modified in all >>
 905      00000 1   <<                         Masters together by DBPUT  to the    >>
 906      00000 1   <<                         associated Detail data set. This     >>
 907      00000 1   <<                         equation is:   4 times each path to  >>
 908      00000 1   <<                         an AUTOmatic master (+) 3 times each >>
 909      00000 1   <<                         path to a MANUAL master (+) 1, the   >>
 910      00000 1   <<                         entry being deleted.                 >>
 911      00000 1
 912      00000 1
 913      00000 1   << The size of the table is defined as follows:                 >>
 914      00000 1
 915      00000 1   DEFINE
 916      00000 1      WORDS'PER'ENTRY   =  4#,
 917      00000 1
 918      00000 1
 919      00000 1      ILR'BLOCKTABSIZE = ILR'MAXBLKENTRYS * WORDS'PER'ENTRY #;
 920      00000 1
 921      00000 1
```

```
 923      00000 1   << The INDEX portion of the ILR BLOCK TABLE is as follows:  >>
 924      00000 1
 925      00000 1   << ********************************************************* >>
 926      00000 1   << *   Free     *          *          *          *          * >>
 927      00000 1   << *   Space    * set 1    * set 2    *   ----   * set x    * >>
 928      00000 1   << *   Pointer  *          *          *          *          * >>
 929      00000 1   << ********************************************************* >>
 930      00000 1
 931      00000 1
 932      00000 1   << The Free Space Pointer is a simple pointer to the next   >>
 933      00000 1   <<    available four (4) word chunk of space.  All entries  >>
 934      00000 1   <<    for a given set can be examined by running down the   >>
 935      00000 1   <<    singly linked list.  The pointer for this singly      >>
 936      00000 1   <<    linked list is the first word of the four word entry. >>
 937      00000 1   <<    The end of this list is determined when a zero is     >>
 938      00000 1   <<    found in the ILR'NEXTONE pointer.                     >>
 939      00000 1
 940      00000 1   << The SET indexes are the pointers for each possible set   >>
 941      00000 1   <<    in the database for that sets first four (4) word     >>
 942      00000 1   <<    entry or zero if no entries are present for that set. >>
 943      00000 1
 944      00000 1   DEFINE ILR'BLKNEXTFREE = 0#,
 945      00000 1          ILR'BLKINDEXLEN = DSETCT + 1#;  << 1 for Free Space Pointer >>
 946      00000 1
 947      00000 1
 948      00000 1   INTEGER POINTER ILR'DSETINDEX = ILR'BLOCK'TABLE;
                           DB+014
 949      00000 1
 950      00000 1
 951      00000 1
 952      00000 1
```



```
 954      00000 1   <<   Each four (4) word entry in this table is defined below:  >>
 955      00000 1   <<                                                             >>
 956      00000 1   <<   ********************************************************* >>
 957      00000 1   <<   *                 *                                   *   >>
 958      00000 1   <<   * Next Entry      * Flushed? T/F * Dset Block Number (2 W)  * >>
 959      00000 1   <<   *                 *                                   *   >>
 960      00000 1   <<   ********************************************************* >>
 961      00000 1
 962      00000 1
 963      00000 1   EQUATE   OSDS = 0,  << Off-Set to the pointer to the next entry >>
 964      00000 1            OSBN = 1;  << Off-Set to dset's Block Number (DOUBLE!) >>
 965      00000 1            OSOD = 1;  << Off-Set to On Disc, (flushed to disc ? ) >>
 966      00000 1
 967      00000 1
 968      00000 1   INTEGER POINTER  ILR'INTINDEX = ILR'CURRENTENTRY;
                           DB+257
 969      00000 1   LOGICAL POINTER  ILR'LOGINDEX = ILR'CURRENTENTRY;
                           DB+257
 970      00000 1   DOUBLE  POINTER  ILR'DUBINDEX = ILR'CURRENTENTRY;
                           DB+257
 971      00000 1
 972      00000 1
 973      00000 1   << The next defines are for ease of addressing the block table. >>
 974      00000 1
 975      00000 1   DEFINE    ILR'NEXTONE = ILR'INTINDEX(OSDS) #,
 976      00000 1             ILR'FLUSHED = ILR'LOGINDEX(OSOD) #,
 977      00000 1             ILR'BLKNUM  = ILR'DUBINDEX(OSBN) #;
 978      00000 1
```

```
  980      00000 1   <<                 ****************************
  981      00000 1                     *                          *
  982      00000 1                     *                          *
  983      00000 1                     *        LOCK AREA         *
  984      00000 1                     *                          *
  985      00000 1                     *                          *
  986      00000 1                     ****************************                    >>
  987      00000 1
  988      00000 1
  989      00000 1
  990      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
  991      00000 1   <<   NEXT COMES THE LOCK AREA.  THE BEGINNING OF THIS AREA IS POINTED
  992      00000 1   TO BY THE GLOBAL DBCB POINTER "LOCKAREA".  >>
  993      00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
  994      00000 1
  995      00000 1   EQUATE INITIALLOCKAREA = 256;          << Initial size of lock area>>
 995.4     00000 1   EQUATE MAX'LOCKAREA = 6144,
 995.5     00000 1
 995.6     00000 1         BIT'MAPSIZE  = 96;          << With chunck size of 4, it  >>
 995.7     00000 1                                     << needs 96 words for BIT MAP >>
 995.8     00000 1   DEFINE TOTAL'BITS=BIT'MAPSIZE*16#; << for 6144 words LOCK AREA.  >>
 995.9     00000 1
  996      00000 1
  997      00000 1
  998      00000 1   <<  This lock area has been changed for MPEV and laster version. >>
  999      00000 1
 1000      00000 1   <<  The MPE procedures OBTAIN and RELEASE now expects 4-word     >>
 1001      00000 1   <<  array (instead of 2-word array in pre-MPEV).  All wait fields>>
 1002      00000 1   <<  are changed from 2 words to 4 words.                         >>
 1003      00000 1   <<                                                             >>
 1004      00000 1   <<  Unfortunately, MPEV is only for HPIB machines.  All smaller  >>
 1005      00000 1   <<  machines must still use MPEIV, which means, IMAGE must run   >>
 1006      00000 1   <<  on both MPEIV and MPEV.                                     >>
 1007      00000 1   <<                                                             >>
 1008      00000 1   <<  The new logical variable PREMPE5, which is set at DBOPEN,    >>
 1009      00000 1   <<  is set to TRUE if the current operating system is pre-MPEV.  >>
 1010      00000 1   <<  If the current system is a pre-MPEV operating system, the    >>
 1011      00000 1   <<  following format for wait fields is used:                    >>
 1012      00000 1   <<             |------------------------|                        >>
 1013      00000 1   <<   Word 1    |PIN# - curnt|  Q count  |                        >>
 1014      00000 1   <<             |------------+-----------|                        >>
 1015      00000 1   <<   Word 2    |   Q tail   |  Q head   |                        >>
 1016      00000 1   <<             |------------+-----------|                        >>
 1017      00000 1   <<   Word 3    |          Unused        |                        >>
 1018      00000 1   <<             |------------------------|                        >>
 1019      00000 1   <<   Word 4    |          Unused        |                        >>
 1020      00000 1   <<             |------------------------|                        >>
 1021      00000 1   <<                                                             >>
 1022      00000 1   <<  If the current system is MPEV of later version, the        >>
 1023      00000 1   <<  following format is used:                                   >>
 1024      00000 1   <<             |------------------------|                        >>
 1025      00000 1   <<   Word 1    | PIN# - current owner   |                        >>
 1026      00000 1   <<             |------------------------|                        >>
 1027      00000 1   <<   Word 2    |      Q   head          |                        >>
 1028      00000 1   <<             |------------------------|                        >>
 1029      00000 1   <<   Word 3    |      Q   tail          |                        >>
 1030      00000 1   <<             |------------------------|                        >>
```



```
 1031      00000 1   <<   Word 4    |      Q   count         |                        >>
 1032      00000 1   <<             |------------------------|                        >>
 1033      00000 1
 1034      00000 1
 1035      00000 1
```

```
1037      00000 1                    << BASE ENTRY (TYPE 0) >>
1038      00000 1
1039      00000 1   << THIS ENTRY HOLDS 'GLOBAL' LOCKING INFORMATION ABOUT THE
1040      00000 1      WHOLE DATABASE. IT ACTS AS THE ROOT OF THE LIST OF SET
1041      00000 1      ENTRIES AND IS POINTED AT BY 'BASEENTRY'.            >>
1042      00000 1
1043      00000 1
1044      00000 1   DEFINE BASEENTRYSIZE=14 #,
1045      00000 1
1046      00000 1   << ************************************ >>
1047      00000 1   << *              TYPE                * >> TYPE =0#,
1048      00000 1   << ************************************ >>
1049      00000 1   << *  LINKED LIST OF OWNED ENTRIES     * >> OWNERLIST =1#,
1050      00000 1   << ************************************ >>
1051      00000 1   << *         OWNERS  USERID           * >> OWNER =2#,
1052      00000 1   << ************************************ >>
1053      00000 1   << *       OWNER   OPEN COUNT         * >> OWNER'OCNT=3#,
1054      00000 1   << ************************************ >>
1055      00000 1   << *   NO OF LOCKS WITHIN DATABASE     * >> BUSY =4#,
1056      00000 1   << ************************************ >>
1057      00000 1   << *                                  * >> BUSYWAIT=5#,
1058      00000 1   << ***                                ***>>
1059      00000 1   << *       4 WORD WAIT FIELD          * >>
1060      00000 1   << ***                                ***>>
1061      00000 1   << *                                  * >>
1062      00000 1   << ***                                ***>>
1063      00000 1   << *                                  * >>
1064      00000 1   << ************************************ >>
1065      00000 1   << *                                  * >> LOCKWAIT=9#,
1066      00000 1   << ***                                ***>>
1067      00000 1   << *       4 WORD WAIT FIELD          * >>
1068      00000 1   << ***                                ***>>
1069      00000 1   << *                                  * >>
1070      00000 1   << ***                                ***>>
1071      00000 1   << *                                  * >>
1072      00000 1   << ************************************ >>
1073      00000 1   << * ROOT OF LINKED LIST OF DATASETS * >> SETLIST =13#;
1074      00000 1   << ************************************ >>
```



```
1076      00000 1                    << SET ENTRY (TYPE 1) >>
1077      00000 1
1078      00000 1   << A SET ENTRY EXISTS FOR EVERY DATASET THAT HAS BEEN
1079      00000 1      REFERRED TO IN A LOCK REQUEST. IT CONTAINS INFORMATION
1080      00000 1      REGARDING THE LOCK STATE OF THE SET ITSELF AND ACTS AS
1081      00000 1      THE ROOT OF THE LIST OF PREDICATES CURRENTLY LOCKED IN
1082      00000 1      THAT SET. SINCE ONLY ONE ITEM NAME CAN BE LOCKED AT A
1083      00000 1      TIME IN A GIVEN SET, THE IDENTITY, TYPE AND LENGTH OF
1084      00000 1      THE ITEM IS KEPT IN THE SET ENTRY ALSO. >>
1085      00000 1
1086      00000 1   DEFINE SETENTRYSIZE =19 #,
1087      00000 1
1088      00000 1   << ************************************                    >>
1089      00000 1   << *              TYPE                *    TYPE           >>
1090      00000 1   << ************************************                    >>
1091      00000 1   << *  LINKED LIST OF OWNED ENTRIES     *    OWNERLIST      >>
1092      00000 1   << ************************************                    >>
1093      00000 1   << *         OWNERS USERID            *    OWNER          >>
1094      00000 1   << ************************************                    >>
1095      00000 1   << *       OWNER  OPEN COUNT          *    OWNER'OCNT      >>
1096      00000 1   << ************************************                     >>
1097      00000 1   << *   NO OF LOCKS WITHIN DATASET      *    BUSY           >>
1098      00000 1   << ************************************                     >>
1099      00000 1   << *                                  *    BUSYWAIT       >>
1100      00000 1   << ****                               ***                  >>
1101      00000 1   << *       4 WORD WAIT FIELD          *                    >>
1102      00000 1   << ***                                ***                  >>
1103      00000 1   << *                                  *                    >>
1104      00000 1   << ***                                ***                  >>
1105      00000 1   << *                                  *                    >>
1106      00000 1   << ************************************                     >>
1107      00000 1   << *                                  *    LOCKWAIT       >>
1108      00000 1   << ***                                ***                  >>
1109      00000 1   << *       4 WORD WAIT FIELD          *                    >>
1110      00000 1   << ***                                ***                  >>
1111      00000 1   << *                                  *                    >>
1112      00000 1   << ***                                ***                  >>
1113      00000 1   << *                                  *                    >>
1114      00000 1   << ************************************                     >>
1115      00000 1   << *    LINKED LIST OF DATASETS       *    SETLIST        >>
1116      00000 1   << ************************************ >>
1117      00000 1   << *         SET NUMBER               * >> SETNO =14#,
1118      00000 1   << ************************************ >>
1119      00000 1   << * ITEM NUMBER CURRENTLY LOCKED      * >> SITEM =15#,
1120      00000 1   << ************************************ >> SITEMDESCRIPTOR =15#,
1121      00000 1   << *     TYPE OF ABOVE ITEM           * >> SITEMTYPE =16#,
1122      00000 1   << ************************************ >>
1123      00000 1   << *   LENGTH OF ABOVE ITEM           * >> SITEMLEN =17#,
1124      00000 1   << ************************************ >>
1125      00000 1   << * ROOT OF LOCKED PREDICATE LIST     * >> LISTROOT =18#;
1126      00000 1   << ************************************ >>
1127      00000 1
1128      00000 1   <<SITEM,SITEMTYPE, AND SITEMLENGTH MUST REMAIN TOGETHER AND IN
1129      00000 1     THAT ORDER, SINCE THEY ARE CONSIDERED A SINGLE 3-WORD ENTITY
1130      00000 1     IN DBLOCK/UNLOCK. >>
```

```
1132      00000 1                  << PREDICATE ENTRY (TYPE 2) >>
1133      00000 1
1134      00000 1   << A PREDICATE ENTRY IS CREATED EACH TIME A REQUEST TO LOCK
1135      00000 1      RECORDS IS MADE. WHEN THE LOCKING PROCESS RELEASES THE LOCK
1136      00000 1      IT IS DESTROYED AND THE SPACE IS RETURNED TO THE FREE POOL.
1137      00000 1      THE ENTRY CONTAINS ENOUGH INFORMATION TO DETERMINE WHETHER
1138      00000 1      OTHER LOCKS CAN BE GRANTED OR NOT. >>
1139      00000 1
1140      00000 1
1141      00000 1   DEFINE PREDOVERHEAD =11 #,
1142      00000 1
1143      00000 1   << ********************************* >>
1144      00000 1   << *           TYPE                * >> << TYPE   >>
1145      00000 1   << ********************************* >>
1146      00000 1   << * LINKED LIST OF OWNED ENTRIES * >> << OWNERLIST   >>
1147      00000 1   << ********************************* >>
1148      00000 1   << *      OWNERS  USERID           * >> << OWNER >>
1149      00000 1   << ********************************* >>
1150      00000 1   << *     OWNER    OPEN COUTN       * >>   <<  OWNER  >>
1151      00000 1   << ********************************* >>
1152      00000 1   << *   DOUBLY-LINKED LIST OF       * >> FWD =4#,
1153      00000 1   << ***                           *** >>
1154      00000 1   << *    ENTRIES WITHIN SET         * >> REV =5#,
1155      00000 1   << ********************************* >>
1156      00000 1   << *                               * >> PWAIT =6#,
1157      00000 1   << ***                           *** >>
1158      00000 1   << *     4 WORD WAIT FIELD         * >>
1159      00000 1   << ***                           *** >>
1160      00000 1   << *                               * >>
1161      00000 1   << ***                           *** >>
1162      00000 1   << *                               * >>
1163      00000 1   << ********************************* >>
1164      00000 1   << * ADDRESS OF PARENT SET ENTRY   * >> PSET =10#,
1165      00000 1   << ********************************* >>
1166      00000 1   << *      PREDICATE RELOP          * >> PRELOP =11#,  BODY =11#,
1167      00000 1   << ********************************* >>
1168      00000 1   << *            VALUE              * >> PVALUE =12#;
1169      00000 1   << *            :                  * >>
1170      00000 1   << *            .                  * >>
1171      00000 1   << *            .                  * >>
1172      00000 1   << *                               * >>
1173      00000 1   << *                               * >>
1174      00000 1   << ********************************* >>
```



```
1176      00000 1                        << ACCESSOR ENTRY >>
1177      00000 1
1178      00000 1   << AN ACCESSOR ENTRY IS CREATED EACH TIME A PROCESS DBOPENS
1179      00000 1      THE DATABASE. SINCE A PROCESS MAY OPEN A GIVEN DATABASE
1180      00000 1      MORE THAN ONCE, THERE MAY BE MORE THAN ONE ACCESSOR ENTRY
1181      00000 1      FOR A PROCESS. SUCH ENTRIES ARE DIFFERENTIATED BY THE
1182      00000 1      'OPEN COUNT' APPENDED TO THE PIN OF THE ACCESSOR. IN ORDER
1183      00000 1      TO FIND A GIVEN ACCESSOR ENTRY, EACH IS ATTACHED TO A LINKED
1184      00000 1      LIST ORIGINATING IN THE PIN HASH TABLE. >>
1185      00000 1
1186      00000 1   DEFINE ACCENTRYSIZE=8#,
1187      00000 1
1188      00000 1   << ********************************** >>
1189      00000 1   << *   LINKED HASH LIST             * >> AHASH =0#,
1190      00000 1   << ********************************** >>
1191      00000 1   << *   PIN -- ACCESSOR ID.          * >> AIDENT=1#,
1192      00000 1   << ********************************** >>
1193      00000 1   << *   OPEN COUNT                   * >> AOPENCNT=2#,
1194      00000 1   << ********************************** >>
1195      00000 1   << *   HEAD OF OWNED ENTRY LIST     * >> OWNERLISTHEAD=3#,
1196      00000 1   << ********************************** >>
1197      00000 1   << *   HEAD OF SORTED PREDICATE CHAIN * >> SORTLISTHEAD=4#,
1198      00000 1   << ********************************** >>
1199      00000 1   << *   ULCB  DST  NO.               * >> ADST =5#,
1200      00000 1   << ********************************** >>
1201      00000 1   << * LOCK * OPEN * WAITCODE     */*D*L* >> FLAGWORD =6#,
1202      00000 1   << ********************************** >>
1203      00000 1
1204      00000 1   << *   POINTER TO WAIT DUBLEWORD    * >> WAITPTR=7#;
1205      00000 1   << ********************************** >>
1206      00000 1
1207      00000 1
1208      00000 1   << NOTE. AHASH MUST REMAIN THE FIRST ENTRY IN THIS TABLE >>
1209      00000 1
1210      00000 1   DEFINE LOCKED'MODE'1OR2 = FLAGWORD).(15:1 #,
1211      00000 1          OPENVIADS       = FLAGWORD).(14:1 #,
1212      00000 1          LOCK'INTENT     = FLAGWORD).(0:4  #,
1213      00000 1                            << 0 NO LOCKS IN PROGRESS >>
1214      00000 1                            << 1 DATABASE LOCK IN PROGRESS >>
1215      00000 1                            << 2 DATA SET LOCK IN PROGRESS >>
1216      00000 1                            << 3 DATA ENTRY LOCK IN PROGRESS >>
1217      00000 1          MODE'OPEN       = FLAGWORD).(4:6  #,
1218      00000 1          WAITREASON      = FLAGWORD).(10:3 #;
1219      00000 1
1220      00000 1   DEFINE BASELOCKWAIT    = 1#,
1221      00000 1          BASEBUSYWAIT    = 2#,
1222      00000 1          SETLOCKWAIT     = 3#,
1223      00000 1          SETBUSYWAIT     = 4#,
1224      00000 1          PREDLOCKWAIT    = 5#;
```

```
1226      00000 1                     << PIN HASH ROOT TABLE >>
1227      00000 1
1228      00000 1   << EACH ACCESSOR ENTRY IS ATTACHED TO A LINKED LIST WHOSE ROOT
1229      00000 1      IS IN THIS TABLE AT AN OFFSET OF PIN MODULO 16 >>
1230      00000 1
1231      00000 1
1232      00000 1
1233      00000 1   DEFINE HASHTABLESIZE = 16#;
1234      00000 1
1235      00000 1   <<              ******************
1236      00000 1                   *        0       *   <- HASHBASE
1237      00000 1                   ******************
1238      00000 1                   *        1       *
1239      00000 1                   ******************
1240      00000 1                   *                *
1241      00000 1                   .                .
1242      00000 1                   .                .
1243      00000 1                   .                .
1244      00000 1                   .                .
1245      00000 1                   .                .
1246      00000 1                   *                *
1247      00000 1                   ******************
1248      00000 1                   *       14       *
1249      00000 1                   ******************
1250      00000 1                   *       15       *
1251      00000 1                   ******************
1252      00000 1   >>
```



```
1254      00000 1                     << WAITING  PROCESS  MAP >>
1255      00000 1
1256      00000 1   << THIS IS A 256-BIT MAP IN WHICH EACH BIT CORRESPONDS TO A PROCESS.
1257      00000 1      THIS BIT IS A ONE IF THE PROCESS HAS BEEN PLACED IN A WAIT STATE
1258      00000 1      BY THE IMAGE LOCKING SYSTEM, ZERO OTHERWISE.  NOTE THE BIT ONLY
1259      00000 1      REFLECTS IMAGE-CREATED WAIT STATES.
1259.34   00000 1
1259.35   00000 1
1259.36   00000 1   NOTE:     This is one bit for each PIN #.  For MPE IV the highest
1259.37   00000 1             PIN # is 256.   For MPE V the highest PIN # is 1024.
1259.38   00000 1
1259.39   00000 1
1260      00000 1   >>
1261      00000 1
1262      00000 1
1263      00000 1   DEFINE WAITMAPSIZE = 64#;
1264      00000 1   <<
1265      00000 1   ************************************
1266      00000 1   * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *
1267      00000 1   * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *  <- POINTER WAITMAP
1268      00000 1   *                 :                *
1269      00000 1   :                 .                :
1270      00000 1   .                 .                .
1271      00000 1   :                 .                :
1272      00000 1   * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *
1273      00000 1   * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *
1274      00000 1   ************************************
1275      00000 1   >>
1276      00000 1
1277      00000 1   DEFINE  IF'WAITING =
1278      00000 1                              ASSEMBLE (DUP);
1279      00000 1                              TOS:=WAITMAP(TOS/16);
1280      00000 1                              X:=B; <<PIN NO>>
1281      00000 1                              ASSEMBLE(TBC 0,X; DDEL);
1282      00000 1                              IF <> #;
1283      00000 1
1284      00000 1   DEFINE SET'WAITING =
1285      00000 1                              ASSEMBLE (DUP);
1286      00000 1                              TOS:=WAITMAP(TOS/16);
1287      00000 1                              X:=B;
1288      00000 1                              ASSEMBLE(TSBC 0,X; XCH);
1289      00000 1                              WAITMAP(TOS/16):=TOS; #;
1290      00000 1
1291      00000 1   DEFINE CLEAR'WAITING =
1292      00000 1                              ASSEMBLE (DUP);
1293      00000 1                              TOS:=WAITMAP(TOS/16);
1294      00000 1                              X:=B;
1295      00000 1                              ASSEMBLE(TRBC 0,X; XCH);
1296      00000 1                              WAITMAP(TOS/16):=TOS; #;
```

```
00000 1                  << INPUT PREDICATES >>
00000 1
00000 1  << THIS IS THE LAYOUT OF THE PROCESSED INPUT PREDICATES THAT
00000 1     ARE STORED IN THE TRAILER AREA PRIOR TO ACTUAL LOCKING>>
00000 1
00000 1  DEFINE
00000 1
00000 1  << *************************************** >>
00000 1  << *  TOTAL LENGTH OF THIS PREDICATE      * >> RLENGTH=0#,
00000 1  << *************************************** >>
00000 1  << *                                      * >>
00000 1  << *                                      * >> RSET=1#,
00000 1  << *               SET                    * >> SORTLINK=2#,
00000 1  << *               NAME                   * >> TYPES=3#,
00000 1  << *                OR                    * >>
00000 1  << *              NUMBER                  * >>
00000 1  << *            (8 WORDS)                 * >>
00000 1  << *                                      * >>
00000 1  << *************************************** >>
00000 1  << *                                      * >>
00000 1  << *                                      * >>RITEM=9#,RITEMDESCRIPTOR=9#,
00000 1  << *               ITEM                   * >>RITEMTYPE=10#,
00000 1  << *               NAME                   * >>RITEMLEN=11#,
00000 1  << *                OR                    * >>
00000 1  << *              NUMBER                  * >>
00000 1  << *            (8 WORDS)                 * >>
00000 1  << *                                      * >>
00000 1  << *************************************** >>
00000 1  << *       RELATIONAL OPERATOR            * >>
00000 1  << *************************************** >> RRELOP=17#,
00000 1  << *                                      * >>
00000 1  << *              VALUE                   * >> RVALUE=18#;
00000 1  << .                :                     * >>
00000 1  << .                .                     . >>
00000 1  << *                                      . >>
00000 1  << *************************************** >>
```



```
00000 1   <<
00000 1                  ******************************
00000 1                  *                            *
00000 1                  *                            *
00000 1                  *           DATA             *
00000 1                  *                            *
00000 1                  *                            *
00000 1                  *                            *
00000 1                  ****                      ****
00000 1                  *                            *
00000 1                  *                            *
00000 1                  *           SET              *
00000 1                  *                            *
00000 1                  *                            *
00000 1                  ****                      ****
00000 1                  *                            *
00000 1                  *                            *
00000 1                  *          BUFFER            *
00000 1                  *                            *
00000 1                  *                            *
00000 1                  ****                      ****
00000 1                  *                            *
00000 1                  *                            *
00000 1                  *           AREA             *
00000 1                  *                            *
00000 1                  *                            *
00000 1                  ******************************
00000 1                                                                >>
00000 1
00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
00000 1  <<    NEXT COMES THE DBCB BUFFER AREA.  THE BEGINNING OF THIS AREA
00000 1  IS POINTED TO BY THE GLOBAL DBCB POINTER "BUFFERAREA".  EACH "TABLE
00000 1  ENTRY" IN THIS AREA CONSISTS OF A DATA SET BUFFER PRECEDED BY
00000 1  SEVERAL WORDS OF INFORMATION REGARDING THE BUFFER'S CONTENTS.
00000 1  (THIS PORTION IS CALLED THE BUFFER HEADER.)  THERE ARE "BUFFERCOUNT"
00000 1  EQUAL-SIZED ENTRIES IN THE BUFFER AREA.  THE DATA BUFFER PORTION
00000 1  OF EACH ENTRY IS OF SIZE "BUFFLGTH", WHICH IS LARGE ENOUGH TO
00000 1  ACCOMODATE A BLOCK OF ANY DATA SET IN THE DATA BASE.  THE BUFFERS
00000 1  ARE MANAGED BY THE IMAGE PROCEDURES SO THAT THEY ARE SHARED BY
00000 1  ALL DATA SETS.  >>
00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
00000 1
00000 1  EQUATE INITIALBUFFERCT = 4;       << INITIAL VALUE OF BUFFERCOUNT >>
00000 1
00000 1
00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
00000 1  <<    THESE DECLARATIONS DEFINE THE FORMAT OF, AND CONTROL ACCESS
00000 1  TO, EACH ENTRY (BUFFER HEADER + DATA BUFFER) OF THE BUFFER AREA. >>
00000 1  <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
00000 1
00000 1  << FIRST, OFFSETS (WORD OR DOUBLEWORD, AS APPROPRIATE) INTO AN ENTRY >>
00000 1
00000 1  EQUATE OB'BUFF'OFFSETS = 0,  << DUMMY TO IDENTIFY THIS JUNK >>
00000 1
00000 1         OBPB    = 0,     << PREVIOUS BUFFER IN QUEUE >>
00000 1         OBNB    = 1,     << NEXT BUFFER IN QUEUE >>
00000 1         OBBN    = 1,     << BLOCK NUMBER >>
```

```
1393    00000 1         OBDS    = 4,     << DATA SET NUMBER >>
1394    00000 1         OBML    = 5,     << MEDIA RECORD LENGTH >>
1395    00000 1         OBBF    = 6,     << BLOCKING FACTOR >>
1396    00000 1         OBBC    = 7,     << BUSY COUNT >>
1397    00000 1         OBDF    = 8,     << DIRTY FLAG >>
1398    00000 1         OBYD    = 9,     << YEAR AND DAY OF MODIFICATION >>
1399    00000 1         OBTIME  = 5;     << TIME OF BUFFER MODIFY (CLOCK)>>
1400    00000 1
1401    00000 1         BUFFERHDRSIZE = 12;  << OVERALL BUFFER HEADER LENGTH >>
1402    00000 1
1403    00000 1
1404    00000 1   << WORD OFFSETS CALCULATED FROM THOSE ABOVE >>
1405    00000 1
1406    00000 1   EQUATE OBBNH    = 2 * OBBN,       << BLOCK NUM, HIGH WORD >>
1407    00000 1         OBBNL    = OBBNH + 1,      << BLOCK NUM, LOW WORD >>
1408    00000 1         OBBNLDS  = OBDS - OBBNL,   << WDS FROM LOW BLK# TO SET# >>
1409    00000 1         OBBNLBNH = OBBNH - OBBNL;  << WDS FROM LOW BLK# TO HIGH BLK# >>
1410    00000 1
1411    00000 1
1412    00000 1   << SYMBOLIC NAMES FOR CONTENTS OF A BUFFER AREA ENTRY.  INTEGER >>
1413    00000 1   <<   POINTER "BUFFERENTRY"  *MUST*  BE SET FOR VALID USE OF     >>
1414    00000 1   <<   THESE SYMBOLS.                                             >>
1415    00000 1
1416    00000 1   INTEGER POINTER INTBUFFERENTRY = BUFFERENTRY;
                        DB+256
1417    00000 1   LOGICAL POINTER LOGBUFFERENTRY = BUFFERENTRY;
                        DB+256
1418    00000 1   DOUBLE  POINTER DUBBUFFERENTRY = BUFFERENTRY;
                        DB+256
1419    00000 1
1420    00000 1   << THE ABOVE DECLARATIONS FORM THE BASIS FOR ALL BUFFERENTRY-
1421    00000 1                 RELATIVE DECLARATIONS.  >>
1422    00000 1
1423    00000 1   DEFINE
1424    00000 1
1425    00000 1   BUFF'PREVBUFFER = INTBUFFERENTRY(OBPB)#, << PREVIOUS BUFFER IN Q >>
1426    00000 1   BUFF'NEXTBUFFER = INTBUFFERENTRY(OBNB)#, << NEXT BUFFER IN QUEUE >>
1427    00000 1   BUFF'BLOCKNUM   = DUBBUFFERENTRY(OBBN)#, << BLOCK NUMBER >>
1428    00000 1   BUFF'DSETNUM    = INTBUFFERENTRY(OBDS)#, << DATA SET NUMBER >>
1429    00000 1   BUFF'MEDIALGTH  = INTBUFFERENTRY(OBML)#, << MEDIA RECORD LENGTH >>
1430    00000 1   BUFF'BLOCKFAC   = INTBUFFERENTRY(OBBF)#, << BLOCKING FACTOR >>
1431    00000 1   BUFF'BUSYCOUNT  = INTBUFFERENTRY(OBBC)#, << BUSY COUNT >>
1432    00000 1   BUFF'DIRTY      = LOGBUFFERENTRY(OBDF)#, << T/F = Dirty/Clean >>
1433    00000 1   BUFF'YEARDAY    = INTBUFFERENTRY(OBYD)#,   << YEAR-DAY OF MODIFY >>
1434    00000 1   BUFF'MODIFYTIME = DUBBUFFERENTRY(OBTIME)#, << HOUR-MIN OF MODIFY >>
1435    00000 1   BUFF'DATABUFFER = INTBUFFERENTRY(BUFFERHDRSIZE)#;
1436    00000 1                                   << ACTUAL DATA BUFFER >>
1437    00000 1
1438    00000 1
1439    00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
1440    00000 1   << END OF DBCB BUFFER AREA >>
1441    00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
```



```
1443    00000 1   <<                  ****************************
1444    00000 1                       *                          *
1445    00000 1                       *       TRAILER AREA       *
1446    00000 1                       *                          *
1447    00000 1                       ****************************
1448    00000 1                       *          ";."            *
1449    00000 1                       ****************************                  >>
1450    00000 1
1451    00000 1
1452    00000 1
1453    00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
1454    00000 1   << IMMEDIATELY BEHIND THE I/O BUFFERS IS THE TRAILER AREA.  IT IS    >>
1455    00000 1   <<   AT LEAST AS LARGE AS THE LARGEST DATA ENTRY IN THE DATA BASE,   >>
1456    00000 1   <<   AND IS NEVER SMALLER THAN 256 WORDS.  IT IS USED BY IMAGE       >>
1457    00000 1   <<   PROCEDURES AS A WORK AREA, AND AS A STAGING AREA FOR TRANSFER   >>
1458    00000 1   <<   OF DATA TO AND FROM THE USER'S STACK.                          >>
1459    00000 1   <<                                                                  >>
1460    00000 1   << THE DBCB SENTINEL IS A 2-BYTE FIELD IMMEDIATELY FOLLOWING THE    >>
1461    00000 1   <<   TRAILER AREA.  IT IS INITIALIZED BY DBOPEN TO ";.".            >>
1462    00000 1   <<  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  >>
1463    00000 1
1464    00000 1
1465    00000 1   <<  *  *  *  *  *  *  *  *  *  *  >>
1466    00000 1   << END OF DATA BASE CONTROL BLOCK >>
1467    00000 1   <<  *  *  *  *  *  *  *  *  *  *  >>
```

```
1469      00000 1   <<               LAYOUT OF USER LOCAL CONTROL BLOCK
1470      00000 1
1471      00000 1
1472      00000 1                      ****************************
1473      00000 1                      *        ULCB HEADER       *
1474      00000 1                      ****************************   **
1475      00000 1                      *    USER FIXED GLOBALS    *    *
1476      00000 1                      ****************************    *
1477      00000 1                      *    ITEM SECURITY TABLE   *    *** ULCB FIXED
1478      00000 1                      ****************************    *     AREA
1479      00000 1                      *    SET SECURITY TABLE    *    *
1480      00000 1                      ****************************   **
1481      00000 1                      *                          *    *
1482      00000 1                      *        USER LOCAL        *    *
1483      00000 1                      *         DATA SET         *    *
1484      00000 1                      *      CONTROL BLOCKS      *    *** ULCB DYNAMIC
1485      00000 1                      *        (ULDSCB'S)        *    *    AREA
1486      00000 1                      *                          *    *
1487      00000 1                      ****************************    *
1488      00000 1                      *   USER DYNAMIC GLOBALS   *    *
1489      00000 1                      ****************************   **
1490      00000 1
1491      00000 1
1492      00000 1                                                                >>
1493      00000 1
1494      00000 1
1495      00000 1   << THE FOLLOWING DECLARATIONS DEFINE THE OFFSETS OF ELEMENTS  >>
1496      00000 1   <<   OF THE ULCB HEADER.                                      >>
1497      00000 1
1498      00000 1   EQUATE ULCB'TAG           = 0,  << SEGMENT TYPE ID: "IMAGE2" >>
1499      00000 1          ULCB'DST           = 3,  << DST NUMBER OF THIS XDS (ULCB) >>
1500      00000 1          ULCB'BASEID        = 4,  << DBCB DST NUMBER & OPEN COUNT >>
1501      00000 1          ULCB'HEADER'LENGTH = 5;
```



```
1503      00000 1   <<-------------------------------------------------------------------->>
1504      00000 1   <<                                                                    >>
1505      00000 1   <<                     LAYOUT OF THE ILCB(*)                          >>
1506      00000 1   <<                                                                    >>
1507      00000 1   <<              ***********************************                   >>
1508      00000 1   <<              *        HEADER RECORD            *                   >>
1509      00000 1   <<              ***********************************                   >>
1510      00000 1   <<              *  DBCB LABEL TABLE (ALL OF IT)   *                   >>
1511      00000 1   <<              ***********************************                   >>
1512      00000 1   <<              *       AREA FOR MOVING           *                   >>
1513      00000 1   <<              *   THE "BEFORE" IMAGES, KNOWN    *                   >>
1514      00000 1   <<              *   AS THE "OLD BLOCKS", BEFORE   *                   >>
1515      00000 1   <<              *   MODIFICATION BY DBPUT OR      *                   >>
1516      00000 1   <<              *   DBDELETE                      *                   >>
1517      00000 1   <<              *       AREA IS BUILT TO THE      *                   >>
1518      00000 1   <<              *   MAXIMUM NUMBER OF BLOCKS      *                   >>
1519      00000 1   <<              *   THAT COULD EVER BE MODIFIED   *                   >>
1520      00000 1   <<              *   INSIDE THE BUFFER AREA OF     *                   >>
1521      00000 1   <<              *   THE DBCB, OR THE MAXIMUM      *                   >>
1522      00000 1   <<              *   NUMBER OF BUFFERS THAT THE    *                   >>
1523      00000 1   <<              *   DBCB COULD ALLOCATE.          *                   >>
1524      00000 1   <<              ***********************************                   >>
1525      00000 1   <<                                                                    >>
1526      00000 1   <<                                                                    >>
1527      00000 1   << This array and equates  should be used only when DB is pointing >>
1528      00000 1   << to the ILCB(*) Extra Data Segment.                                 >>
1529      00000 1   <<                                                                    >>
1530      00000 1   <<-------------------------------------------------------------------->>
1531      00000 1
1532      00000 1   logical array       ILCB  (*)       = DB+0;
                    DB+000
1533      00000 1   byte    array       ILCB'B(*)       = ILCB;
                    DB+000
1534      00000 1
1535      00000 1   integer             ILCB'XDSNUM     = DB+3,
                    DB+003
1536      00000 1                       ILCB'XDSLEN     = DB+4,
                    DB+004
1537      00000 1                       ILCB'NEXTREC    = DB+5;
                    DB+005
1538      00000 1
1539      00000 1   integer pointer     ILCB'BUFFER     = DB+6;
                    DB+006
1540      00000 1
1541      00000 1   integer             ILCB'GLOBAL'END = DB+7;
                    DB+007
1542      00000 1
1543      00000 1   integer array       ILCB'HEAD  (*)  = DB+8;
                    DB+010
1544      00000 1   byte    array       ILCB'HEAD'B(*)  = ILCB'HEAD;
                    DB+010
1545      00000 1   logical array       ILCB'HEAD'L(*)  = ILCB'HEAD;
                    DB+010
1546      00000 1   double  array       ILCB'HEAD'D(*)  = ILCB'HEAD;
                    DB+010
1547      00000 1
```

```
1549      00000 1   EQUATE IL'NAME         =  0,        << Log file name          >>
1550      00000 1          IL'GRP          =  4,        << Log file group         >>
1551      00000 1          IL'ACCT         =  8,        << Log file account       >>
1552      00000 1          IL'CREATEDATE   = 12,        << Date log was created   >>
1553      00000 1          IL'LASTDATE     = 13,        << Date of log last access >>
1554      00000 1          IL'BEGINDATE    = 14,        << Date PUT/DELETE started >>
1555      00000 1          IL'ENDDATE      = 15,        << Date PUT/DELETE ended  >>
1556      00000 1          IL'CREATETIME   =  8,        << Time log was created   >>
1557      00000 1          IL'LASTTIME     =  9,        << Time of log last access >>
1558      00000 1          IL'BEGINTIME    = 10,        << Time PUT/DELETE started >>
1559      00000 1          IL'ENDTIME      = 11,        << Time PUT/DELETE ended  >>
1560      00000 1          IL'LABTBL       = 24,        << Address of label TBL   >>
1561      00000 1          IL'TOTHEADERLEN= 25,         << Len of file head/labels >>
1562      00000 1          IL'ENTRYLENGTH = 26,         << Entry (record) length  >>
1563      00000 1          IL'LOGFIRSTBLK = 27,         << First Rec to write blks >>
1564      00000 1          IL'MAXBLOCKS   = 28,         << Maximum blocks modifible>>
1565      00000 1          IL'BLKTBL      = 29,         << Address of Block Area  >>
1566      00000 1          IL'EXECUTING   = 30,         << Is PUT/DELETE executing >>
1567      00000 1          IL'INTRINPCODE = 31,         << Intrinsic pcode, 407/8 >>
1568      00000 1          IL'DSETINX     = 32,         << Dset # being modified  >>
1569      00000 1
1570      00000 1          << Words 33 thru 49 are unused but may be used >>
1571      00000 1          <<      at a later date without forcing changes >>
1572      00000 1          <<      to existing log files.                  >>
1573      00000 1
1574      00000 1          ILCB'HEADLENGTH = 50;       << Over all header length  >>
```



```
1576      00000 1
1577      00000 1   <<
1578      00000 1
1579      00000 1             layout of Master Data Base Control Block (SDBCB)
1580      00000 1
1581      00000 1                    *************************
1582      00000 1                    *   SDBCB tag - "IMAGE0" *
1583      00000 1                    *************************
1584      00000 1                    *   SDBCB  DST  size    *
1585      00000 1                    *************************
1586      00000 1                    *   SDBCB  DST   no.    *
1587      00000 1                    *************************
1588      00000 1                    *                       *
1589      00000 1                    *   SDBCB wait field    *
1590      00000 1                    *       (4 words)       *
1591      00000 1                    *                       *
1592      00000 1                    *************************
1593      00000 1                    *   Max entry count     *
1594      00000 1                    *************************
1595      00000 1                    *   Entry   count       *
1596      00000 1                    *      *********        *
1597      00000 1                    *                       *
1598      00000 1                    *       Entries         *
1599      00000 1                    *          .            *
1600      00000 1                    *          .            *
1601      00000 1                    *          .            *
1602      00000 1                    *          .            *
1603      00000 1                    *************************
1604      00000 1                                                          >>
1605      00000 1   <<                                                     >>
1606      00000 1   <<  Each entry is 1 word long.  It consists of a DST# of     >>
1607      00000 1   <<  an opened database.                                 >>
1608      00000 1   <<                                                     >>
1609      00000 1   <<  This area should only be used when DB is pointing to     >>
1610      00000 1   <<  SDBCB extra data segment.                          >>
1611      00000 1
1612      00000 1
1613      00000 1   LOGICAL ARRAY          SDBCB'TAG(*)=DB+0;
                          DB+000
1614      00000 1   BYTE    ARRAY          SDBCB'TAG'B(*)=SDBCB'TAG;
                          DB+000
1615      00000 1
1616      00000 1   INTEGER                SDBCB'XDSLENGTH=  DB+3;  << Size of the XDS. >>
                          DB+003
1617      00000 1   INTEGER                SDBCB'DSTNUM=DB+4;
                          DB+004
1618      00000 1   ARRAY                  SDBCB'WAIT(*)=DB+5;     << 4-word wait field>>
                          DB+005
1619      00000 1   INTEGER                SDBCB'MAXECNT=DB+9;
                          DB+011
1620      00000 1
1621      00000 1   INTEGER ARRAY          SDBCB'DST(*) =DB+10;   << 1st word is count. >>
                          DB+012
1622      00000 1
1623      00000 1
1624      00000 1   DEFINE                 SDBCB'ENTRYCNT=SDBCB'DST(0) #;
```

Module 3.1

Establishing and Terminating Access Paths for the Data Base

Module 3.2

Module Outline

A. DBOPEN

0. Introduction
1. Open the Root File
2. Examine the Current Use Information
3. Create the DBCB if Current # Users = 0
4. Create the ULCB
5. Set Up the ULCB
6. Update Root File Label 0
7. Set Up PCBX Entry for User's Access to the DBCB
8. Prepare to Exit

B. DBCLOSE

0. Introduction
1. DBCB Preparation
2. Determine Access Mode
3. Mode 3/Rewind the Data Set
4. Mode 2/Close the Data Set
5. Mode 1/Close a Data Base
6. Prepare to Exit Mode 1

Quiz

□ DBOPEN

□ Notes

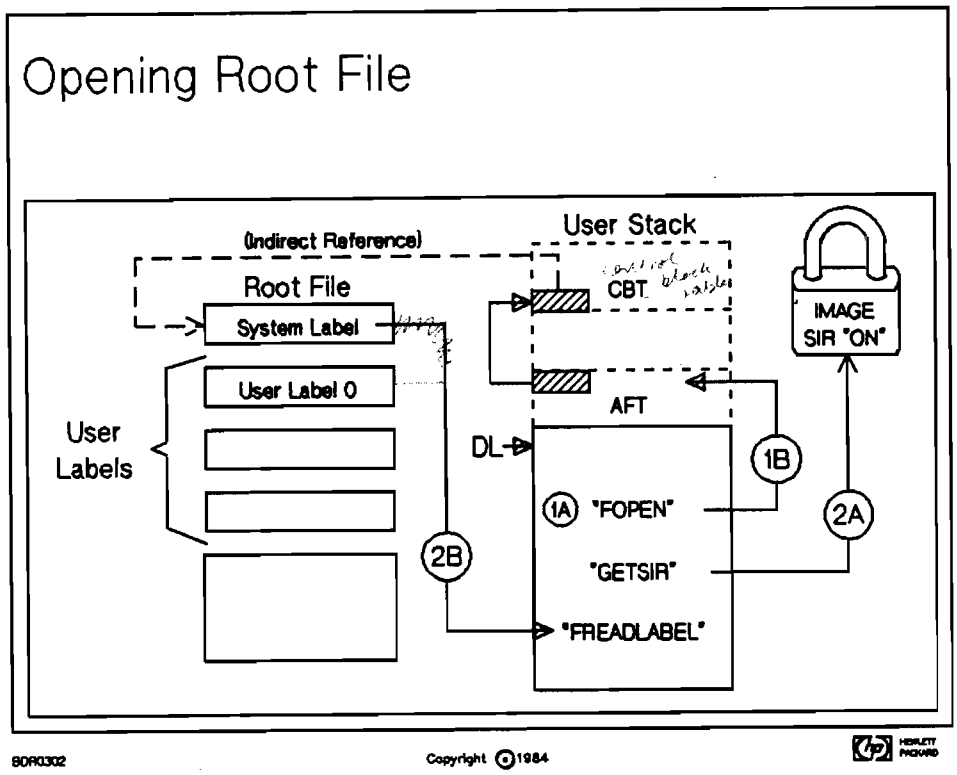
Establishing an Access Path

DBOPEN (Base,Password,Mode,Status)

* Opens the Root File- does not open data sets
* May create the DBCB (first user)
* Creates the ULCB
* Determines access mode for the path
* Determines security restrictions for the path
* Determines other allowable concurrent access modes
* Determines the number of data buffers within the DBCB

□ DBOPEN

□ Notes



1. Open the Root File

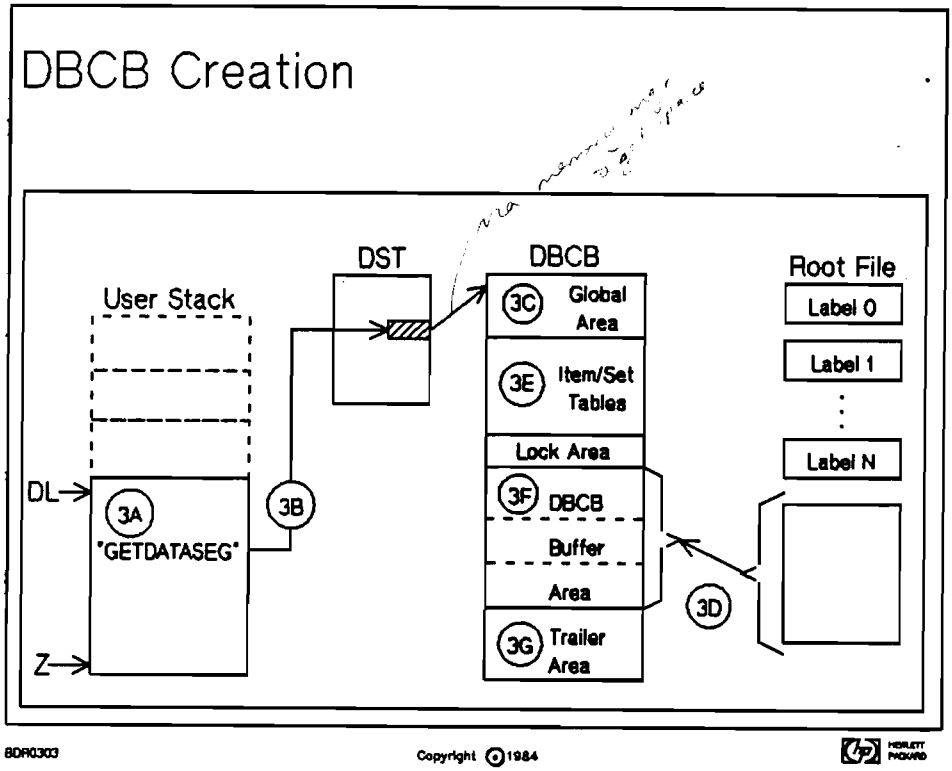
A. Call standard MPE FOPEN.
B. Set up AFT and CBT entry.

2. Examine the Current Use Information

A. Call GETSIR to protect Label 0.
B. Read Label 0 and determine:
- Whether or not there has been a system crash.
- Current number of users.

□ DBOPEN

□ Notes

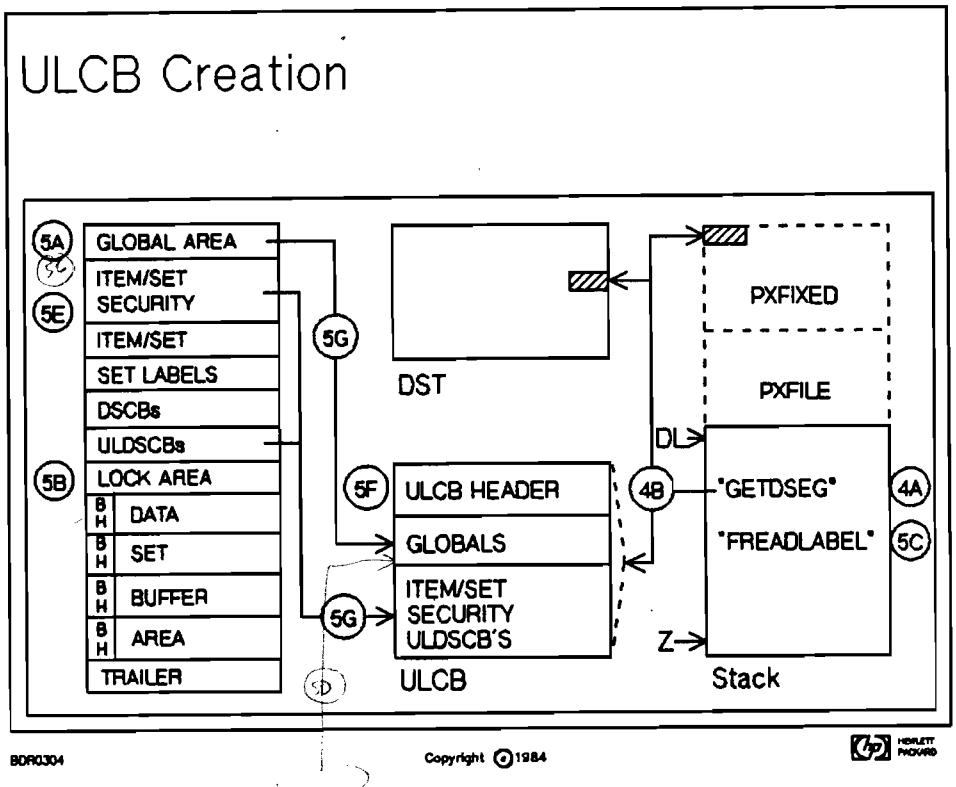


3. Create the DBCB if Currrent # Users = 0

A. Determine the size of the DBCB using the first words of Record 0.
B. Call GETDATASEG to create the initial DBCB.
   - Sets up a DST entry.
   - Does not set up PCBX entry for the user.
C. Read the root file into the DBCB buffer area (may need to expand DBCB).
D. Set up and initialize pointers in the global area.
E. Set up and initilaize all item and set tables.
F. Initialize the buffer area.
   - Zero out each buffer.
   - Set up the buffer header and queue.
G. Set and zero out the trailer area (initially 256w).

□ DBOPEN

□ Notes



4. Create the ULCB

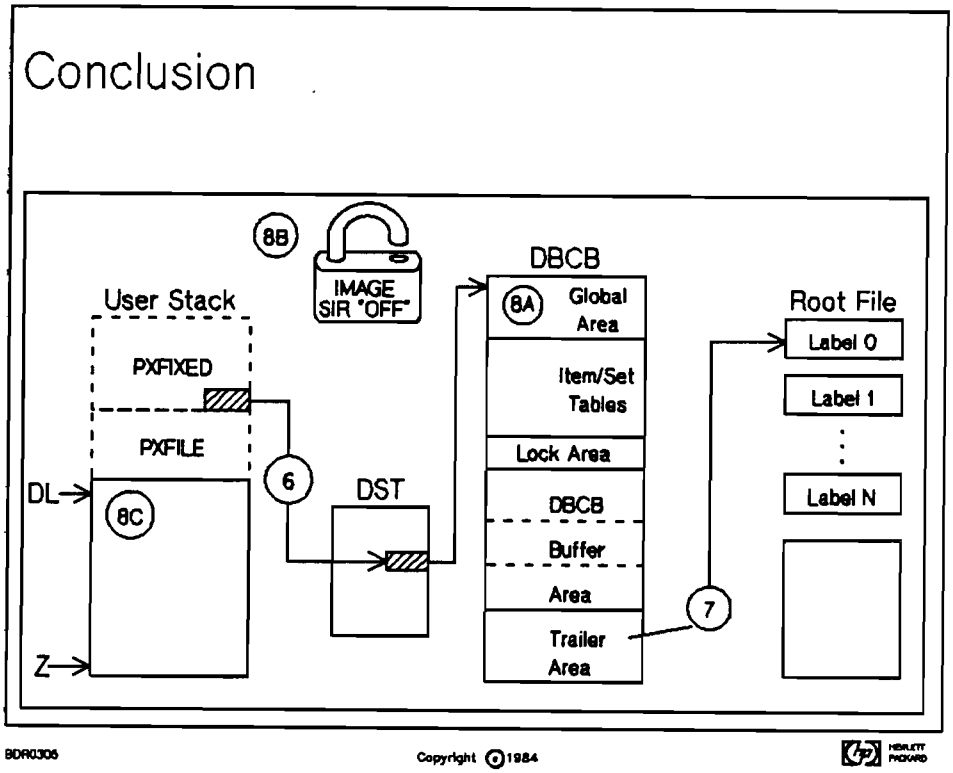
A. Determine ULCB size.
B. Call GETDSEG to create the ULCB.
   - Creates a PCBX entry for the ULCB.
   - Sets up a DST entry.

5. Set up the ULCB

A. Lock the DBCB.
B. Set up user accessor entry.
C. Set up user information.
   - Base-ID = open count + DST #.
   - Set ULDSCB's to zeroes.
   - Set AOptions for data sets (for later FOPENS) and access flags in user fixed globals.
D. Read labels 1 and 2 of root file to determine user class number.
E. Set up the user's item and set security table based on class number entry (user Labels 3 to N).
F. Set up ULCB header.
G. Move user information to the ULCB.

□ DBOPEN

□ Notes



6. Set up PCBX for user's access to the DBCB (Pxfixed)

A. # current users.
B. # of users for each access mode.
C. Set buffer specifications (may cause DBCB expansion).
D. DST # of DBCB.
E. Write Label 0 to disc.

7. Update Root File Label 0

8. Prepare to exit.

A. Unlock the DBCB.
B. Release the IMAGE SIR.
C. Return information to the user.
   - Base-ID.
   - ULCB and DBCB size.

Module 3.8

□ DBCLOSE

□ Notes

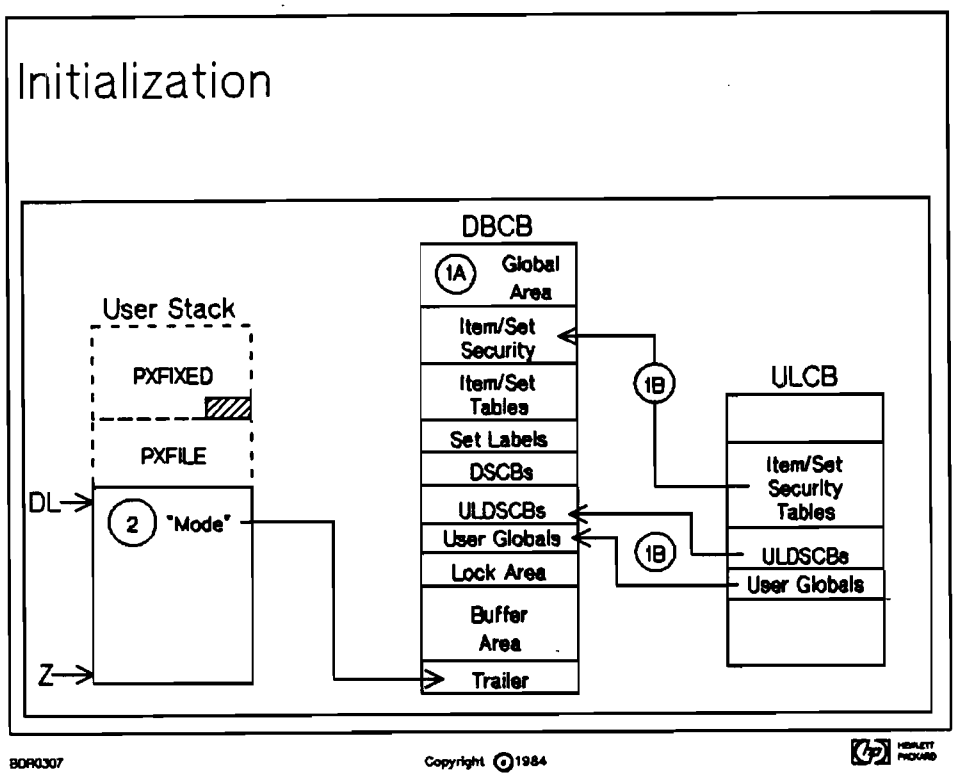
Terminating or Altering Access Paths

DBCLOSE (Base,Dset,Mode,Status)

* Can be used to close the data base
* Can be used to close a data set
* Can be used to rewind a data set
* Closes the root file
* Posts any dirty buffers/labels
* May delete the DBCB (last user)
* Deletes the ULCB
* May adjust the # of buffers

□ DBCLOSE

□ Notes



1. DBCB preparation

(slide 2.83)

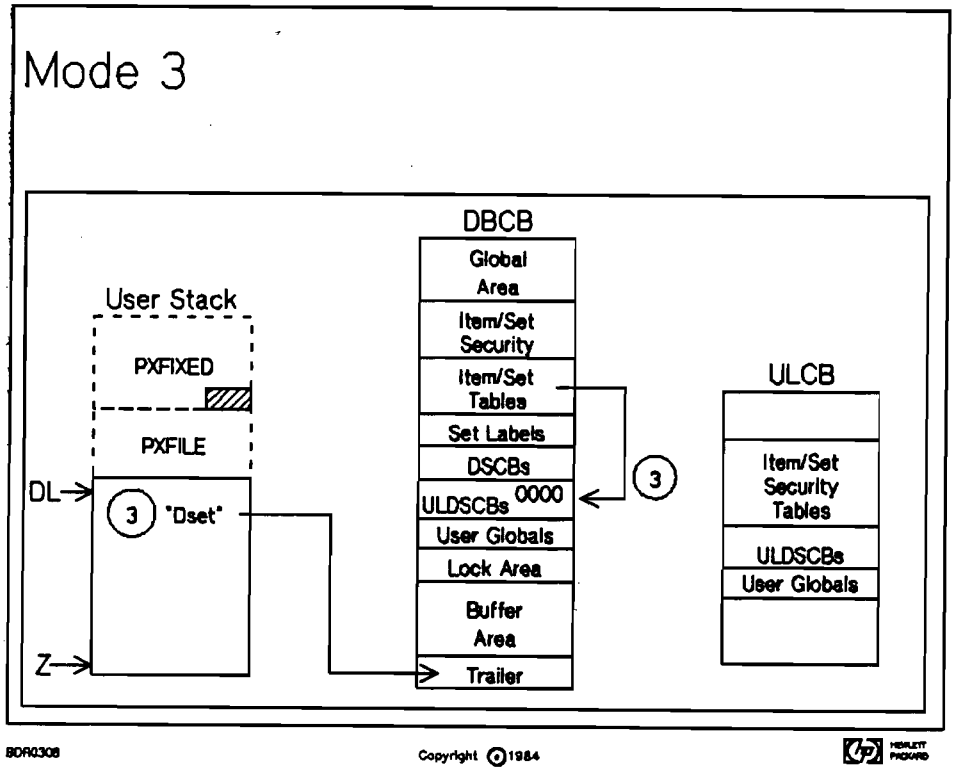
A. Lock the DBCB.
B. Move user information to the DBCB.

2. Determine DBCLOSE mode

A. Mode = 3 Go to step 3.
B. Mode = 2 Go to step 4.
C. Mode = 1 Go to step 5.

□ DBCLOSE

□ Notes

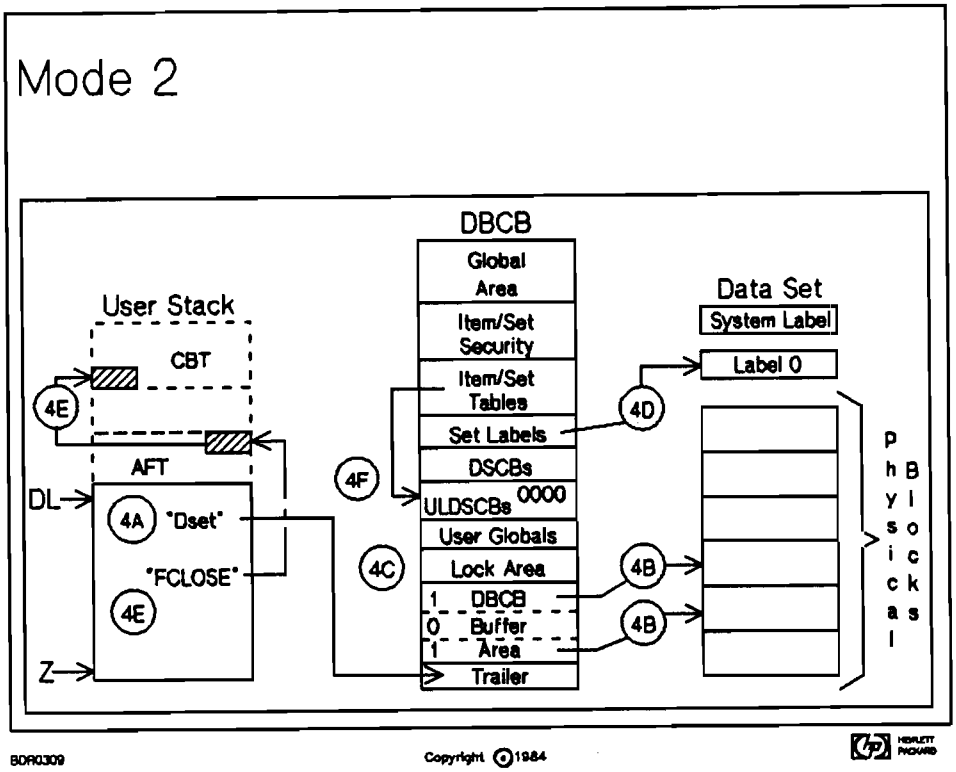


3. Mode = 3 (Rewind the Data Set)

A. Set ULDSCB to zeroes.
B. Prepare to exit.

□ DBCLOSE

□ Notes

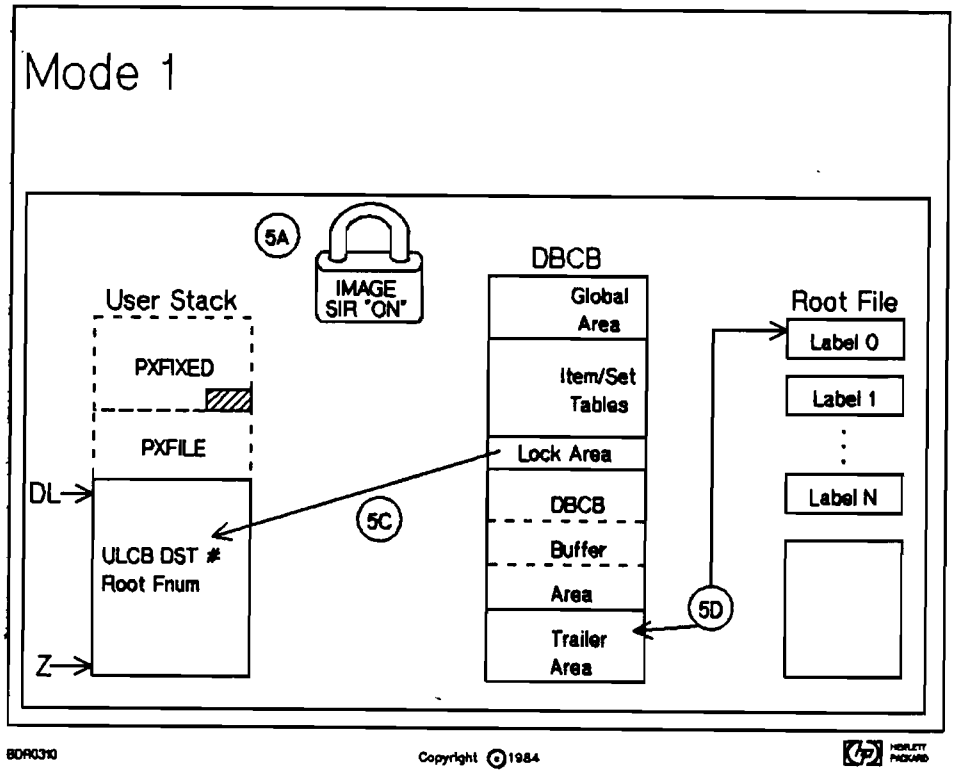


4. Mode 2 (Close a Data Set)

A. Move DSET parameter to trailer and set pointers.
B. Write any "dirty" buffers for the set to disc.
C. Does not release any locks held by user for the set if mode = 2.
D. Write user Label 0 for the set if label is "dirty".
E. Call FCLOSE on the set.
F. Set ULDSCB to zeroes for the set.
G. Prepare to exit.

□ DBCLOSE

□ Notes

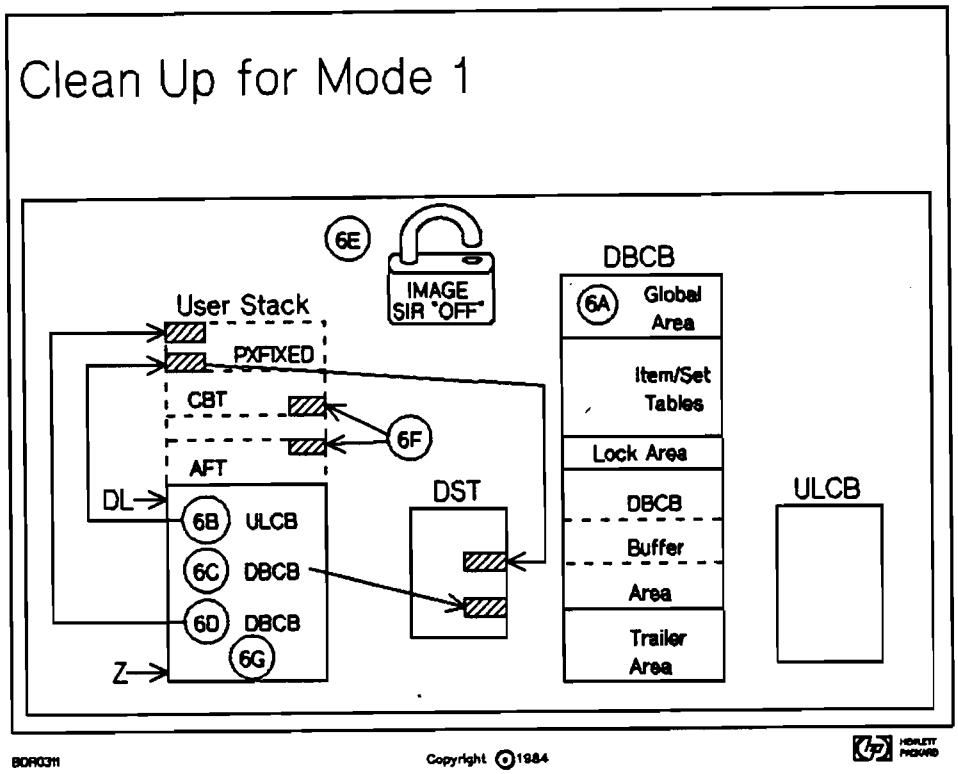


5. Mode 1 (Close the Data Base)

A. GETSIR on IMAGE SIR.
B. Perform Step 4 for all open data sets.
   - Releases all locks on data sets.
C. Prepare DBCB for close.
   - Read accessor entry.
   - Save the root file # and ULCB DST #.
D. Update root file Label 0.
   - Read Label 0.
   - Release accessor entry.
   - If last user, zero out DBCB DST # in Label 0.
   - ~~Adjust # of buffers if necessary (shrink DBCB).~~
   - Write Label 0.
E. Adjust # of buffers if necessary.

□ DBCLOSE

□ Notes



6. Prepare to exit (Mode 1)

A. Unlock the DBCB (check wait queue).
B. Release the ULCB.
C. If user count = 0, release the DST entry for the DBCB.
D. Release the PCBX entry for the DBCB.
E. Release the IMAGE SIR (if mode = 1).
F. Call FCLOSE on the root file.
G. Return base = 0 to the user.

Module 3.15

QUIZ

1. Why does IMAGE use a SIR? to protect label 0 of Rootfile

2. What is the function of GETDATASEG? GETDSEG?

3a. Can the number of buffers change when a user calls DBOPEN? Explain.

3b. Does this cause overhead? How? Yes due to virtual memory

4. When creating the DBCB, how does DBOPEN know how much space to allocate?

5. When a ~~data set~~ root file is opened, where is the AFT number returned?

6. What mode of DBCLOSE would be used to reset pointers? 3
Why this mode and not another?

7. If you close a single data set, are the locks released? NO

8. At DBCLOSE on the data base, can the number of buffers be changed?

9. Is the ULCB released by DBCLOSE mode one (1)? DBCB?

10. Does DBCLOSE use SIRs? Why or why not?

Module 3.16

LAB MGR.EXAMPLE

1. [illegible]

2. Open EXAMPL mode 1

3. /DEBUG

4. Find VLCB (via pin table hash root table)
DB + 201 in DBCB

'IMAGE 2' tag

DDA dst# + offset , A O

DR > PCB = pin#

DBDRIVER
!B = EXAMPL
!M = 1
!@ ;
0

4904 wds DBCB 1st person
5420 wds
5936 wds 5th person

elapsed time = 470

Base id = %2167 Class = 64 DBCB = 5420 VLCB = 72

/DEBUG
? DR
ST = 4?301 , X = 1 , DL = 171644 , Q = [illegible] , Z = 10311 , [illegible]
PCB = 102 , [illegible]

Pin # = %102 %102 / %20 = %2 2nd entry in pin hash root table

DST# of DBCB is 10001/110 111 = %167
(10 bits of Base ID)

?DDA 167 + 201 %667 → pin hash root table
?DDA 167 + 667 1Ø %750 (wd 2, – one word entries & wanted 2nd entry in Pin hash table
↓
Accessor Table

?DDA 167 + 750, 7

000 000502 000 000 000703 000100 000

101 000 080

opt cnt

= %102 [1st entry on the linked list]

ULCB DST# = %203

?IDA 203, 10, A

IMAGE2
0 1 2

wd. 3 = %703 DST# ULCB

wd. 4 = %2[illegible] Base [illegible]

wd [illegible] = %[illegible] open cnt [illegible]

DBDRIVER

This utility runs in non-privileged mode allowing the user to interactively call data base intrinsics. It executes the command and returns the time in milliseconds that it took for the intrinsic to execute.

Parameters can be set defining the data base name, password, mode, etc. There are 4 types of DBDRIVER commands:

* Set parameter values (!)
* Display parameter contents (?)
* Call IMAGE intrinsics (command code)
* Special commands (/)

Parameter Values

SET	INQUIRE	
!B=	?B=	data base name
!Q=		password
!M=	?M=	mode
	?S=	status
!L=	?L=	list of item names
!E=	?E=	buffer contents
!A=	?A=	argument value
!Q=	?Q=	data set name
!I=	?I=	item name

Calling IMAGE Intrinsics

NOTE: Set parameter values *before* specifying intrinsic command.

```
O - DBOPEN     (base,  password, mode, status)
C - DBCLOSE    (base,  dset,     mode, status)
I - DBINFO     (base, qualifier, mode, status, buffer)
F - DBFIND     (base, dset,      mode, status, item,          arg)
G - DBGET      (base, dset,      mode, status, list, buffer, arg)
P - DBPUT      (base, dset,      mode, status, list, buffer)
D - DBDELETE   (base, dset,      mode, status)
U - DBUPDATE   (base, dset,      mode, status, list, buffer)
L - DBLOCK     (base, dset,      mode, status)
R - DBUNLOCK   (base, dset,      mode, status)
K - DBCONTROL  (base, qualifier, mode, status)
X - DBEXPLAIN                          (status)
E - DBERROR                            (status, buffer, length)
B - DBBEGIN    (base, text,      mode, status, textlen)
N - DBEND      (base, text,      mode, status, textlen)
M - DBMEMO     (base, text,      mode, status, textlen)
```

Special Commands

/V	Displays version, update, and fix numbers
/D	Enters debug
/S	Enters system debug (requires Privileged Mode)
/EXIT	Exit
/M	Monitor-- sends all terminal activity to printer (LP)
/W	Turns off monitor mode
//text	Comments
/C	Displays Code Segment Number tables of IMAGE procedures
/R n1,n2,n3	Turn on repeat and execute command n1 times n2 adding n3 to each word in the buffer each time.
/O	Turn off the repeat option.
/PRED p1:p2:p3:p4	Sets a lock descriptor for mode 5,6 locking: p1- data set name in quotes or number p2- data item name in quotes or number p3- relational operator in quotes p4- value in ASCII in quotes
/NOPRED	Erase all previous descriptors that have been set with the /PRED command.

Sample Run

Here is the schema for the *TEST31* data base:

```
BEGIN DATA BASE TEST31;

PASSWORDS:
    1  READER;
    2  WRITER;

ITEMS:
  ONE,     X6;
  TWO,     P12;
  THREE,   I;
  FOUR,    U6;
  FIVE,    R2;
```

Module 4.1

Reading
Data Set Entries

Module 4.2

Module Outline

A. DBFIND

0. Introduction
1. DBCB Preparation
2. Access the Detail Data Set
3. Determine the Appropriate Path for the Detail
4. Access the Master Data Set
5. Determine the Location of the Record in the Master
6. Put the Desired Block into a Buffer
7. Determine if Master Record has a Matching Search Item

Quiz

B. DBGET

0. Introduction
1. DBCB Preparation
2. Prepare for Data Set Access
3. Set Up and Verify the List in the Trailer
4. Mode 1/Re-Read
5. Mode 2/Forward Serial Read
6. Mode 4/Directed Read
7. Mode 5/Forward Chained Read
8. Mode 7/Calculated Read

Worksession

□ DBFIND

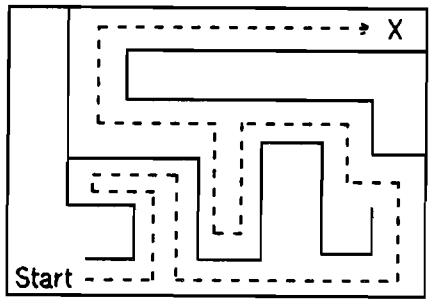
□ Notes

Finding the Beginning of a Detail Chain

DBFIND (Base,Mode,Status,Item,Arg)

* Sets path number within data set

* Sets forward & backward chain pointers for data set (master chain head)



* Does NOT return any data to user

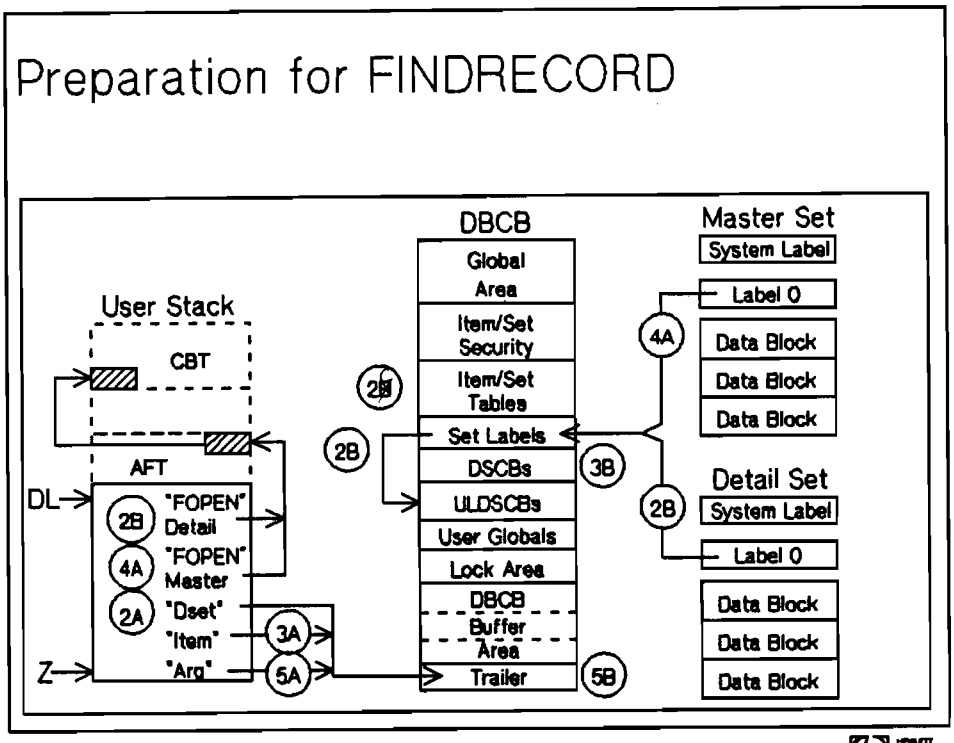
* Detail chain itself may be empty

Module 4.4

□ DBFIND Base, Dset, Mode, Status, Num, Rec

□ Notes

Preparation for FINDRECORD



BDR0402 Copyright ©1984 HEWLETT PACKARD

1. DBCB preparation (2 33)

2. Access the detail data set

A. Read the DSET parameter into the trailer.
B. May require FOPEN. — AFT → CBT on stack.
C. Verify set is a detail. — Set table

3. Determine the appropriate path for the detail

A. Read the item parameter into the trailer.
B. Determine the set number & path number of related master. DSCB is checked.

4. Access the master data set using data set number from 3B

A. May require FOPEN

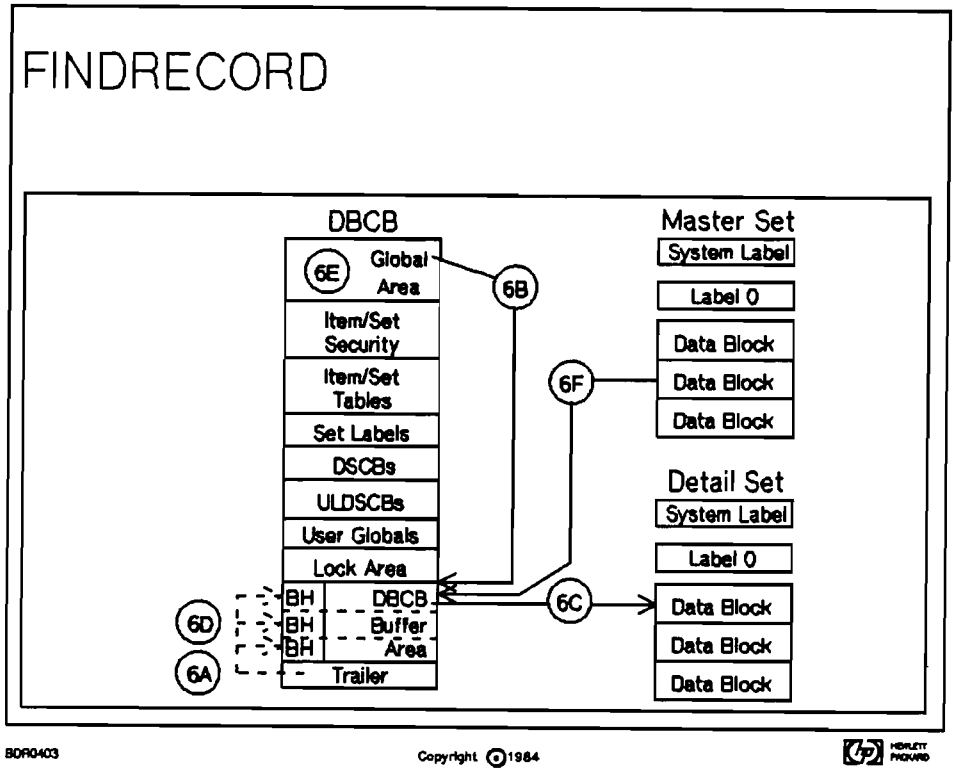
5. Determine the location of the record in the master

A. Read the argument parameter into the trailer.
B. Hash the argument in the trailer.
C. Convert the hash result to a block and offset value (RECORDNAME).

RRN = block# + offset
#records/block

□ DBFIND

□ Notes

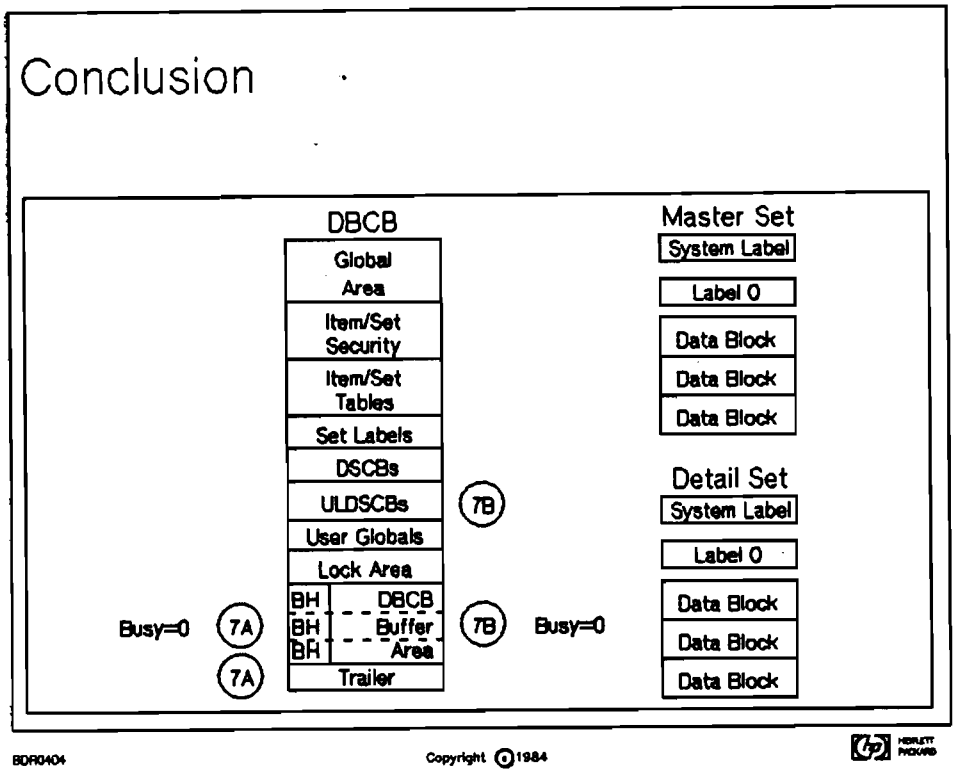


6. Put the desired block into a buffer (FINDRECORD)

A. Checks buffer headers to see if "recordname" is already present (if true go to 6G).
B. Get the first available buffer (DBCB global area).
C. If the dirty flag is set (header) write the buffer to disc.
D. Set up the buffer header.
   - Block #, set #.
   - Media record size and block factor.
E. Scratch pointers are set (DBCB global area).
F. Read the block into the current buffer using FREADDIR.
G. Set the busy flag within the buffer (header).

□ DBFIND

□ Notes



7. Determine if master record has a matching search item

A. If argument ≠ item value and forward ptr ≠ 0:
   - Save forward synonym pointer in current record.
   - Release the buffer (busy = 0).
   - Determine the RECORDNAME.

B. If argument = item value set up the ULDSCB for the detail data set.
   - Current record and path.
   - Chain length, backward and forward pointers.
   - Release the buffer (busy = 0).

8. Prepare to exit

QUIZ

1. What is the basic function of FINDRECORD? to set up a record in a DBCB [illegible] if it is not already there.

2. What effect can long synonym chains in a master have on DBFIND? can cause a DBFIND to take a long time due to I/O - paging [illegible]

□ DBGET

□ Notes

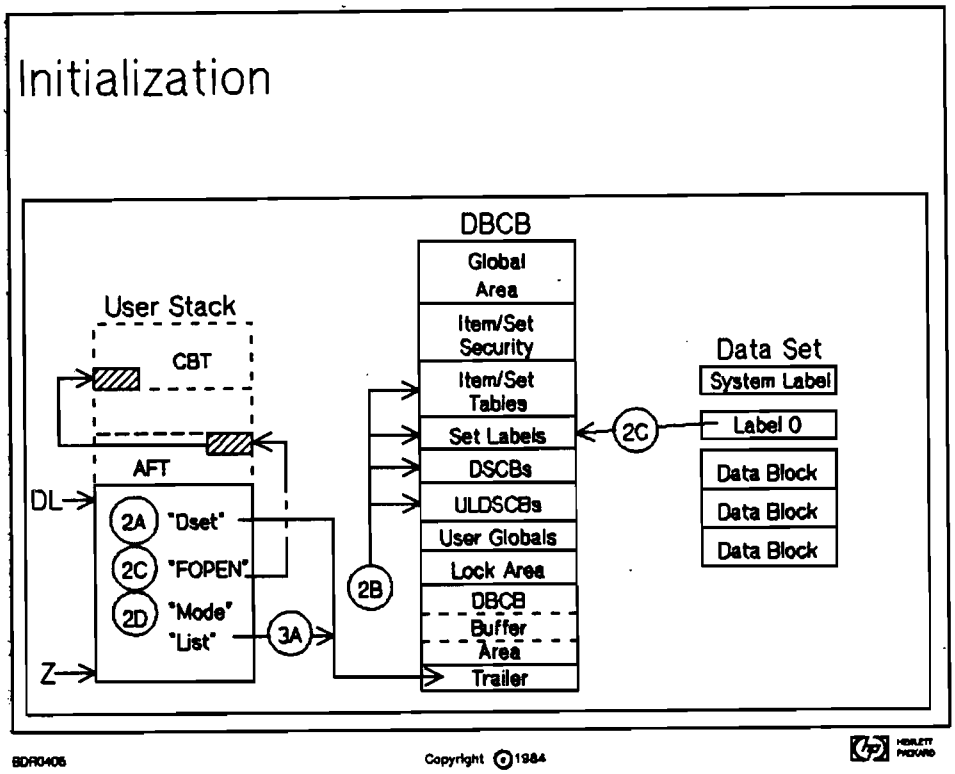
Reading a Data Entry

DBGET (Base,Dset,Mode,Status,List,Buffer,Arg)

* Retrieves the List for the current record in the specified Dset
* The Mode parameter determines how the current record pointer is set
* Opens data sets if necessary

□ DBGET

□ Notes



1. DBCB preparation

2. Prepare for data set access

A. Move DSET parameter to the trailer.
B. Set pointers to data set areas.
C. Call FOPEN on the data set if not opened.
   - Read user Label 0 for the data set.
D. Determine mode.

3. Set up and verify the list in the trailer

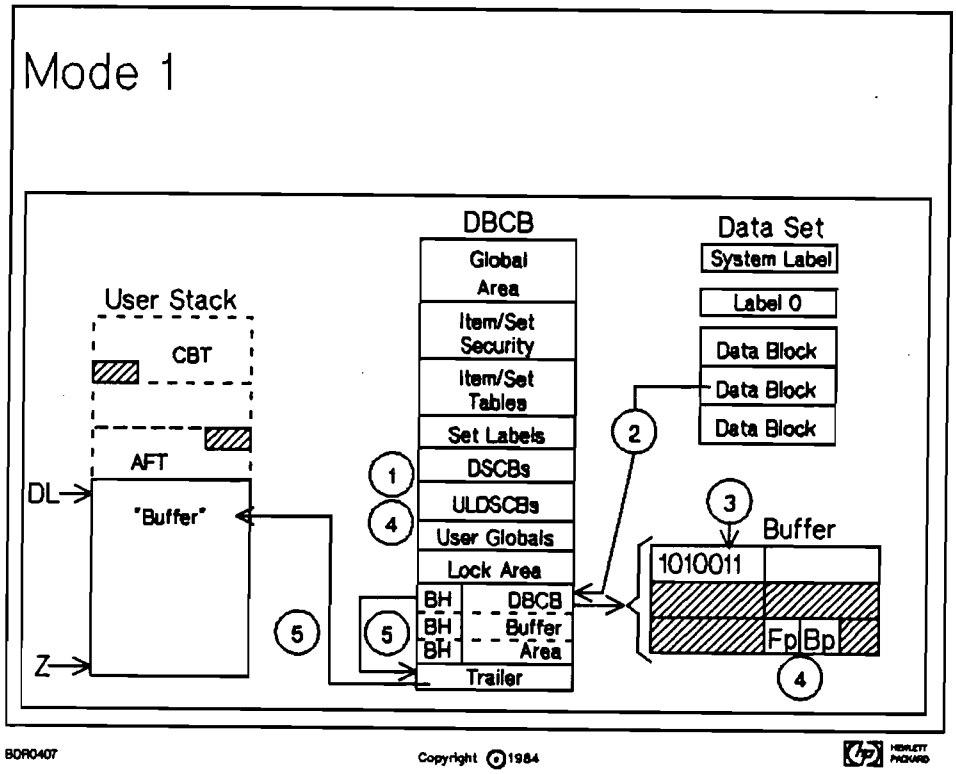
~~the list in the trailer~~

A. Move the list parameter to the trailer.
B. Verify read capability on each item.
   - If write capability on the set, no further checking is needed (set security).
   - If read capability on the set, check read capability on each item.
   - If "*" specified use the current list for the set (ULDSCB).

Module 4.11

☐ DBGET

☐ Notes

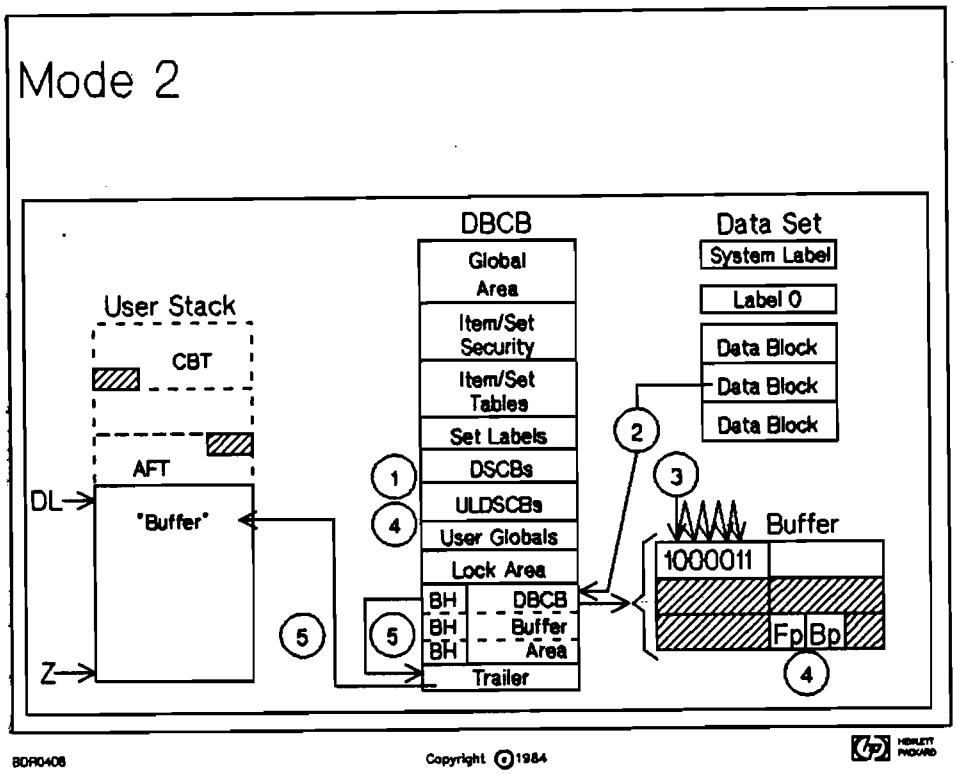


Mode 1 (Re-Read)

1. Determine the current record (ULDSCB) (If = 0 Return).
2. Call FINDRECORD on current RECORDNAME.
3. Check the bit map to see if the requested entry is in the block (If not - Return).
4. Set forward and backward pointers and chain length.
5. Move requested items to the user's stack.
6. Prepare to exit.

□ DBGET

□ Notes



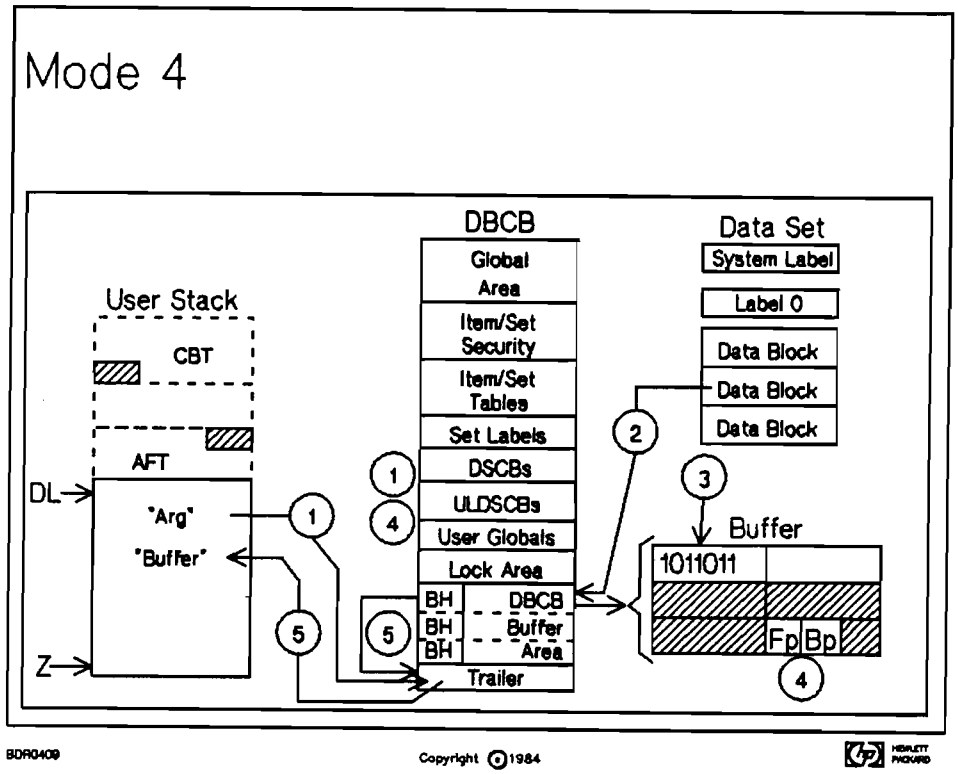
Mode 2 (Forward Serial Read)

1. Advance the current record pointer.
   A. Increment record name by 1 (bounds check).
2. Call FINDRECORD on current record.
3. See if the requested entry is in the block.
   A. Check current record corresponding bit map entry.
      - If = 1 Go to Step 4 (match).
      - If = 0 and is not the last block entry
        Increment the current record by 1
        Check bounds
        Go to Step 3A.
      - If = 0 and is the last entry in the block
        Go to Step 1.
4. Set forward and backward pointers and chain length in the ULDSCB.
5. Move the requested items to the user's stack.
6. Prepare to exit.

Module 4.13

□ DBGET

□ Notes

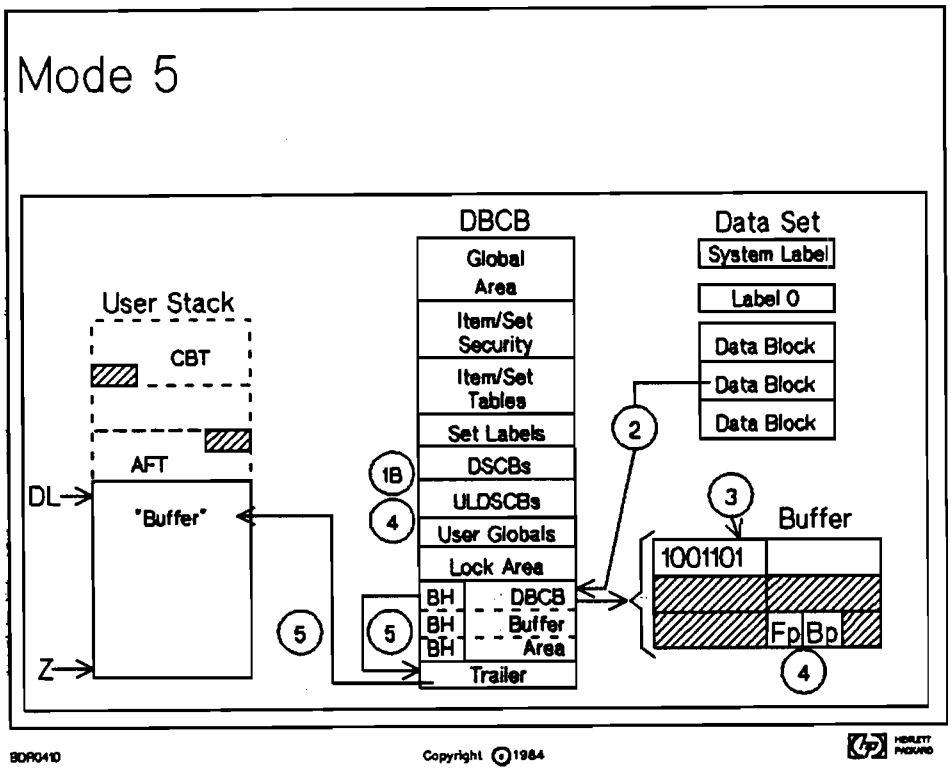


Mode 4 (Directed Read)

1. Set the current record pointer.
   A. Move argument to trailer.
   B. Convert to RECORDNAME format.
   C. Check bound.
2. Call FINDRECORD using RECORDNAME.
3. Check the bit map to see if the requested entry is in the block.
   A. If = 0 set chain information to 0 and return.
   B. If = 1 check search item.
4. Set forward and backward pointers and chain length.
5. Move the requested items to the user's stack.
6. Prepare to exit.

□ DBGET

□ Notes

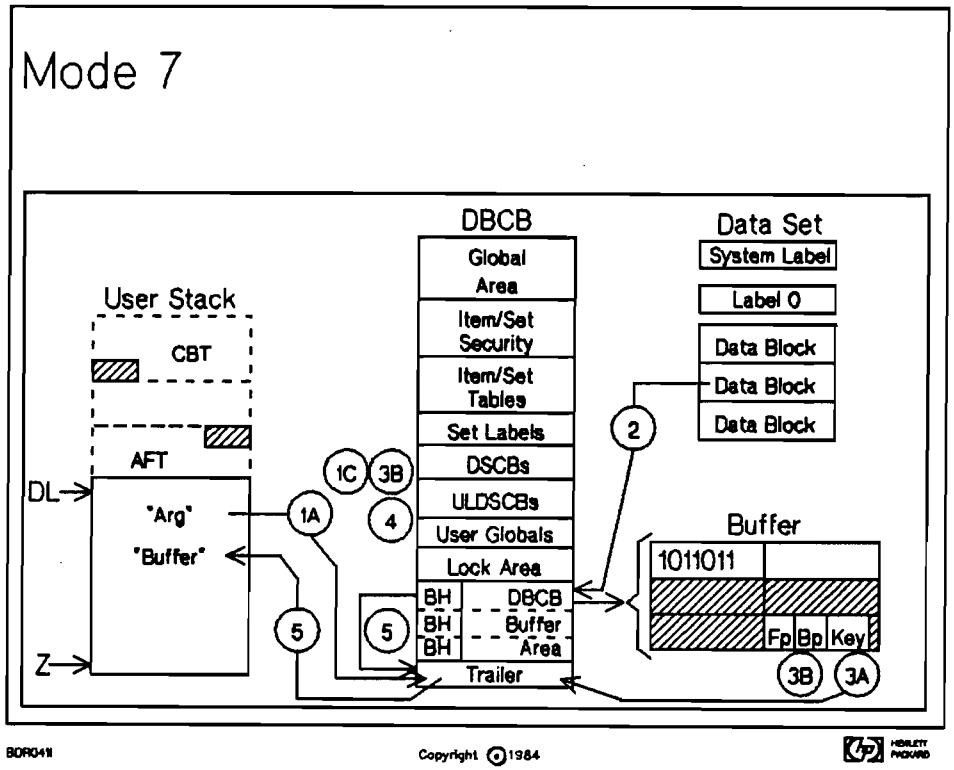


Mode 5 (Forward Chained Read)

1. Advance the current record pointer (ULDSCB).
   A. If forward pointer = 0 return.
   B. Set current RECORDNAME = forward pointer.
2. Call FINDRECORD using current RECORDNAME.
3. Check the bit map to see
   if the requested entry is present.
   A. If = 0 set chain information = 0 and return.
4. Set forward and backward pointers and chain length.
5. Move requested items to the user's stack.
6. Prepare to exit.

□ DBGET

□ Notes



Mode 7 (Calculated Read)

1. Determine the current record pointer.
   A. Move argument to the trailer.
   B. Hash the argument in the trailer.
   C. Determine the 'RECORDNAME' format.
2. Call FINDRECORD using RECORDNAME.
3. Determine if a match has been found.
   A. If argument = record key value go to Step 4.
   B. If argument ≠ record key value
      - If forward synonym pointer = 0 reutrn.
      - if forward synonym pointer ≠ 0
        set current record to forward pointer and go to Step 2.
4. Set forward and backward pointers and chain length.
5. Move the requested items to the user's stack.
6. Prepare to exit.

Worksession

1. For each of the procedure calls determine the minimum number of data blocks or labels which would need to be read if the data base was optimally designed for that particular procedure.

2. Next, identify the situations which could exist causing more than this minimum to be read.

3. Finally, suggest some guidelines to minimize the overheads you mention in Step 2.

Module 4.17

	Minimum # of calls to FINDRECORD (blocks and labels).	Could there be more calls? Explain.	Design guidelines to minimize # of disc I/O's.
DBFIND	**Master Data Set:** 1	**Master Data Set:**	**Master Data Set:**
	Detail Data Set: 0	**Detail Data Set:**	**Detail Data Set:**
DBGET Mode 1 (re-read)	**Data Set:** 1	**Data Set:** 1	**Data Set:**

	Minimum # of calls to FINDRECORD (blocks and labels).	Could there be more calls? Explain.	Design guidelines to minimize # of disc I/O's.
DBGET Modes 2 & 3 (serial)	**Data Set:** 1	**Data Set:** yes if the record is not in the block	**Data Set:** large blocks (ADAGER compress)
DBGET Mode 4 (directed)	**Data Set:** 1	**Data Set:** No	**Data Set:** lots of buffers [illegible]

FIND [illegible]

[illegible] — finds KRIL for checking [illegible]

Module 4.19

	Minimum # of calls to FINDRECORD (blocks and labels).	Could there be more calls? Explain.	Design guidelines to minimize # of disc I/O's.
DBGET Modes 5 & 6 (chained)	**Master Data Set:** 0	**Master Data Set:**	**Master Data Set:**
	Detail Data Set: 1	**Detail Data Set:** No	**Detail Data Set:**
DBGET Mode 7 & ~~8~~ (calculated)	**Master Data Set:** 1	**Master Data Set:**	**Master Data Set:**
	Detail Data Set: 0	**Detail Data Set:**	**Detail Data Set:**

Module 4.20

Module 5.1

Modifying Data Set Entries

Module 5.2

Module Outline

DBUPDATE

0. Introduction
1. DBCB Preparation
2. Check for Update Capability
3. Check for Locking if Opened in Mode 1
4. Set all DBCB Pointers for Data Set
5. Verify the Readability of List Parameter Items
6. Call FINDRECORD Using Current Record Name
7. Check for Record Locking
8. Update the Record
9. Prepare to Exit

DBDELETE

0. Introduction
1. DBCB Preparation
2. Verify DSET Parameter and Access
3. Determine DSET Type
    a. Detail Delete
    b. Master Delete

DBPUT

0. Introduction
1. DBCB Preparation
2. Verify DSET Parameter and Access
3. Prepare for Data Set Access
4. Validate the List Parameter
5. Check Label 0
6. Determine DSET Type
    a. Master
    b. Detail/Sorted Chains

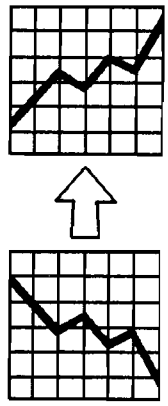
Worksession

□ DBUPDATE

□ Notes

Updating a Data Entry

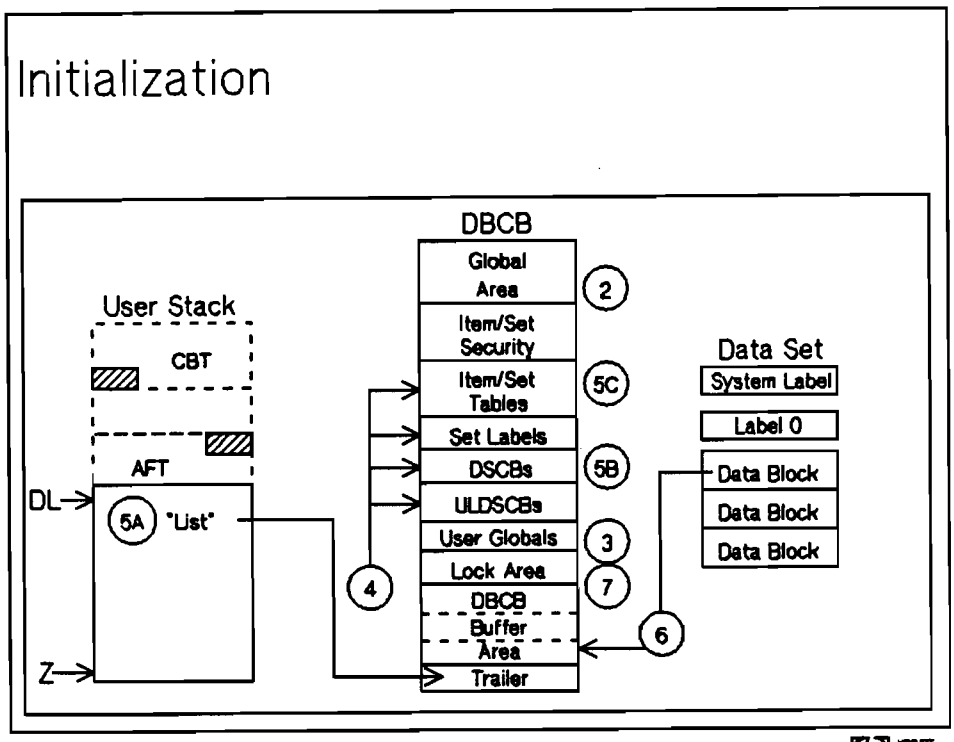
DBUPDATE (Base,Dset,Mode,Status,List,Buffer)



* Updates the current record in Dset
* Modifies the value of items in the List to values in Buffer
* Search items can not be modified
* If DB opened in Mode 1 a data entry lock must cover the entry before and after the update (lock value)
* Current record, forward & backward ptrs are unchanged

□ DBUPDATE

□ Notes



1. DBCB preparation

2. Check for update capability

(Update Flag in user fixed globals)

3. Check for locking

- If opened in Mode 1 at the base or set level.

4. Set all DBCB pointers for the given data set

5. Verify the readability of items in the list parameter

A. Move list to trailer.
B. Verify the item is in the set (DSCB).
C. Verify the user has write access to the set or read access to each item.

6. Call FINDRECORD

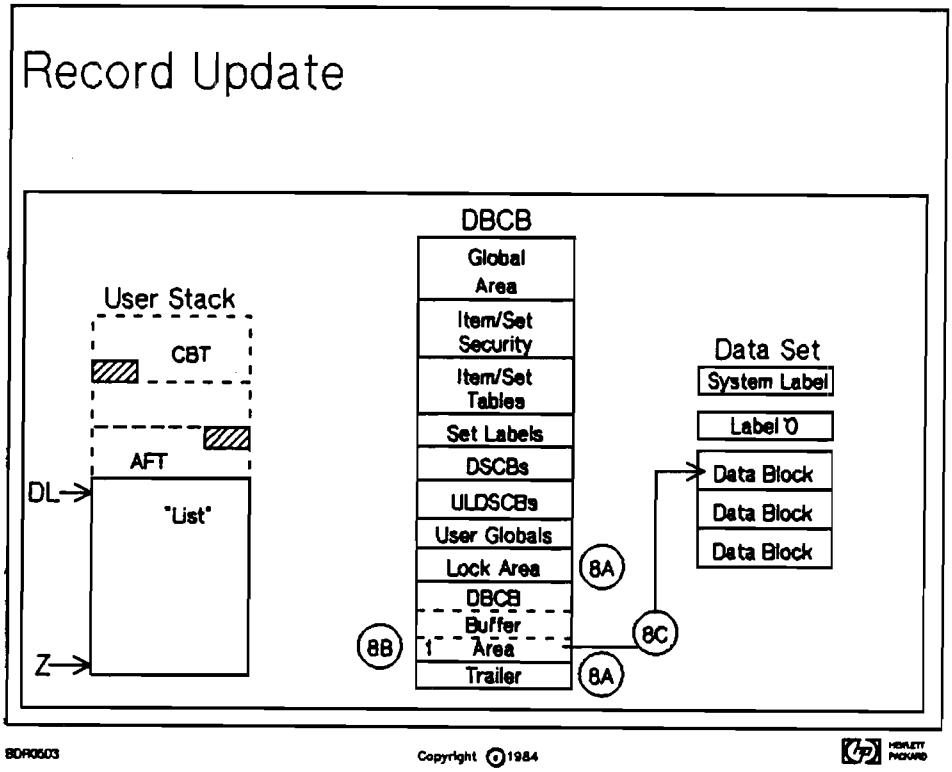
- Use the current record name.

7. Check for record locking

- If base or set is NOT locked.

□ DBUPDATE

□ Notes



8. Update the record

A. Compare the value of each item in the trailer list to the value of each item in the media record, if different...
   - Check for a search item.
   - Check item security (if only read access at set level).
   - If the item is a lock item, check that its new value is locked.
   - Move the new value to the buffer.

B. Mark the buffer dirty and release it.

C. Write all dirty buffers to disc.

Prepare to exit

□ DBDELETE

□ Notes

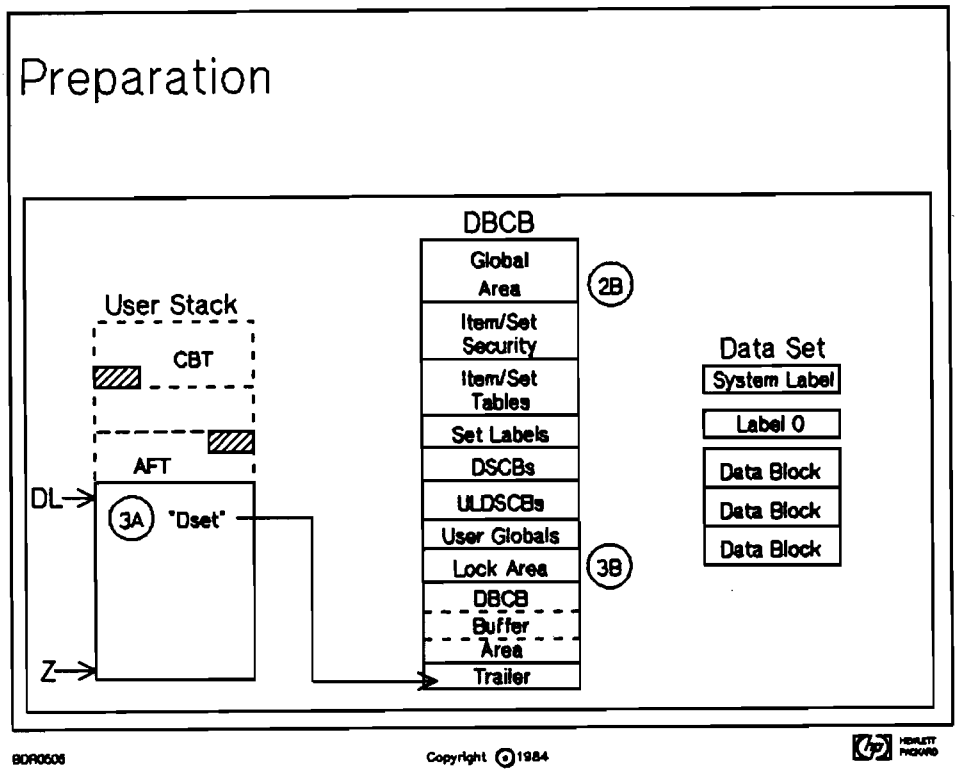
Deleting Data Entries

DBDELETE (Base,Dset,Mode,Status)

* Deletes the current entry for Dset
* All detail chains must be empty for a successful DBDELETE on a master
* Handles detail chain maintenance for DBDELETE on a detail
* Deletes automatic master if the last entry on corresponding detail chain is deleted
* If DB is opened in Mode 1, data set or base must be locked

□ DBDELETE

□ Notes



1. DBCB preparation

2. Verify DSET parameter and access

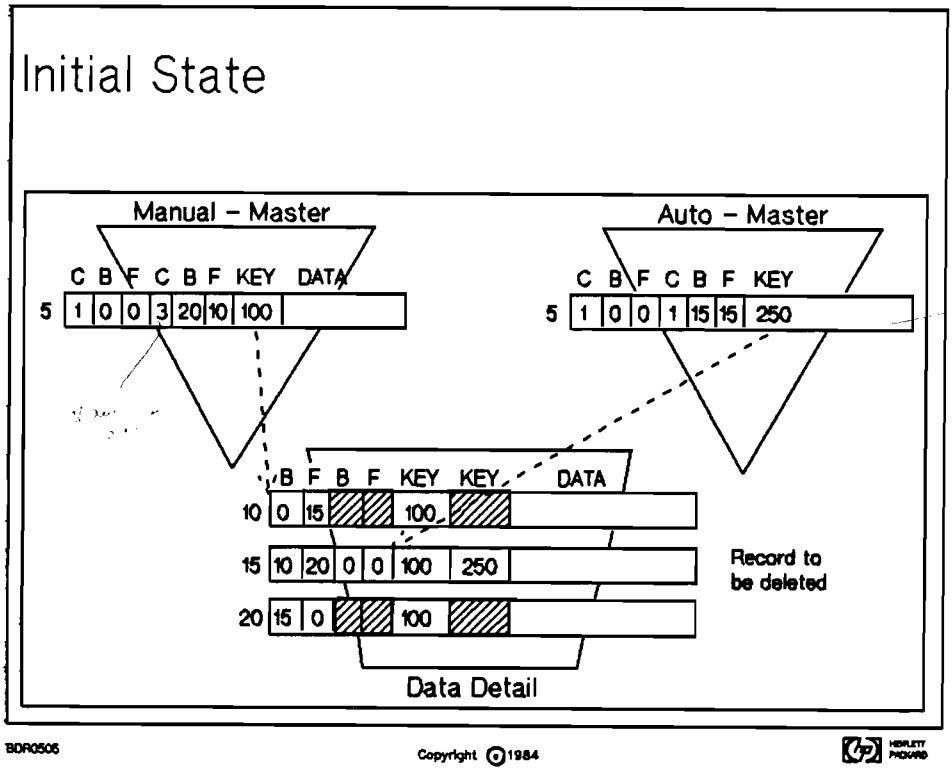
A. Move DSET to the trailer.
B. Modify flag must be on.
C. Check for locking if DBOPEN Mode = 1.

3. If DSET type = "A" or "M"
Go to master delete

4. If DSET type = "D"
Go to detail delete

□ DBDELETE

□ Notes

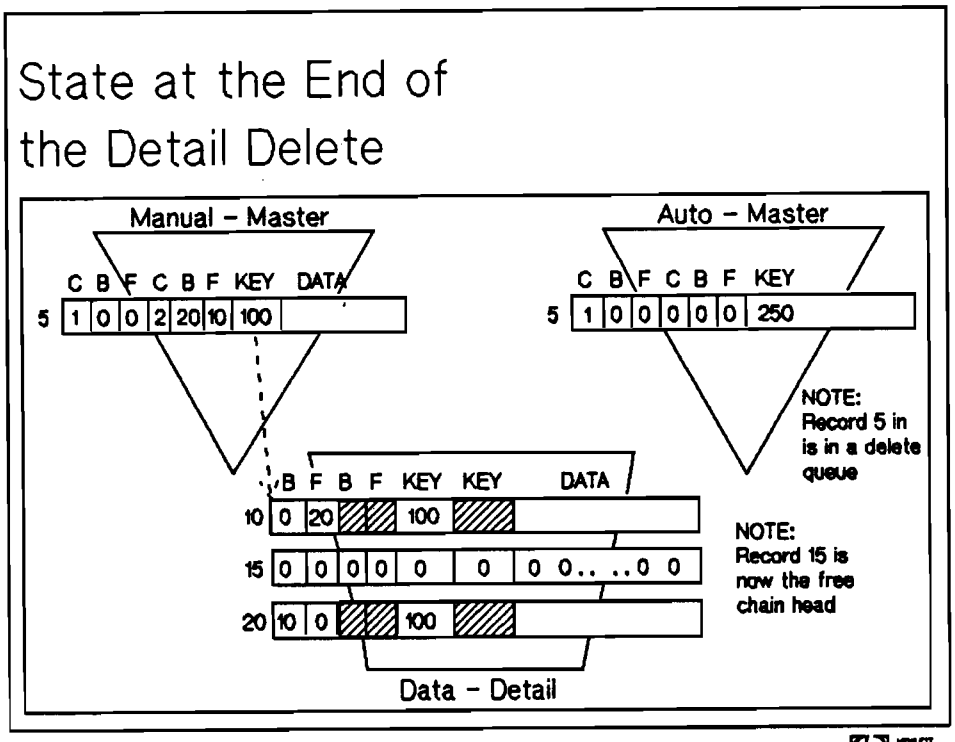


Detail

1. Modify all master data set chain heads (counts and pointers).
   - A. Call FINDRECORD on detail entry to be deleted (current record).
   - B. Check path table. For each path to a master:
     - Hash the key in the detail record to find the master chain head (call FINDRECORD).
     - May have to chase a synonym chain until the key is found.
     - Decrement the chain count.
     - Modify forward and backward pointers only if first or last entry on the chain is deleted.
     - Check chain head counts for automatic masters, if = 0 put the entry in the MASTERS-TO-BE-DELETED queue.

□ DBDELETE

□ Notes

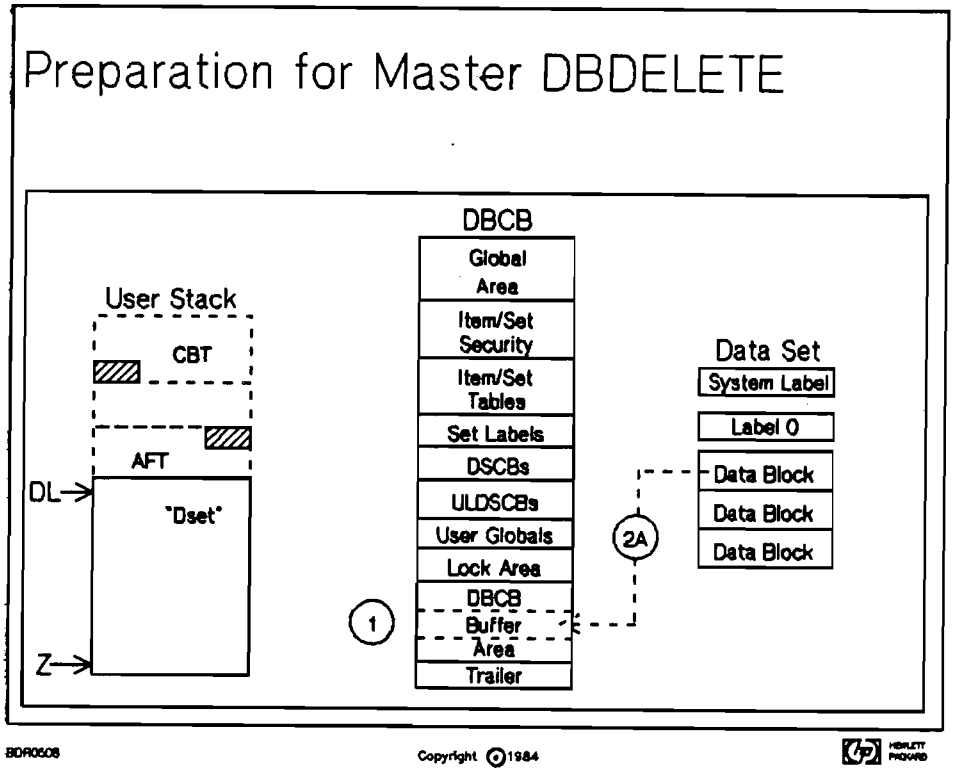


Detail (cont.)

2. Delete the detail
   A. For each path:
      - Call FINDRECORD on forward and backward pointers.
      - Update the chain link.
   B. Set bit map to 0 and zero the media record.
   C. Put the entry on the free chain head.

□ DBDELETE

□ Notes



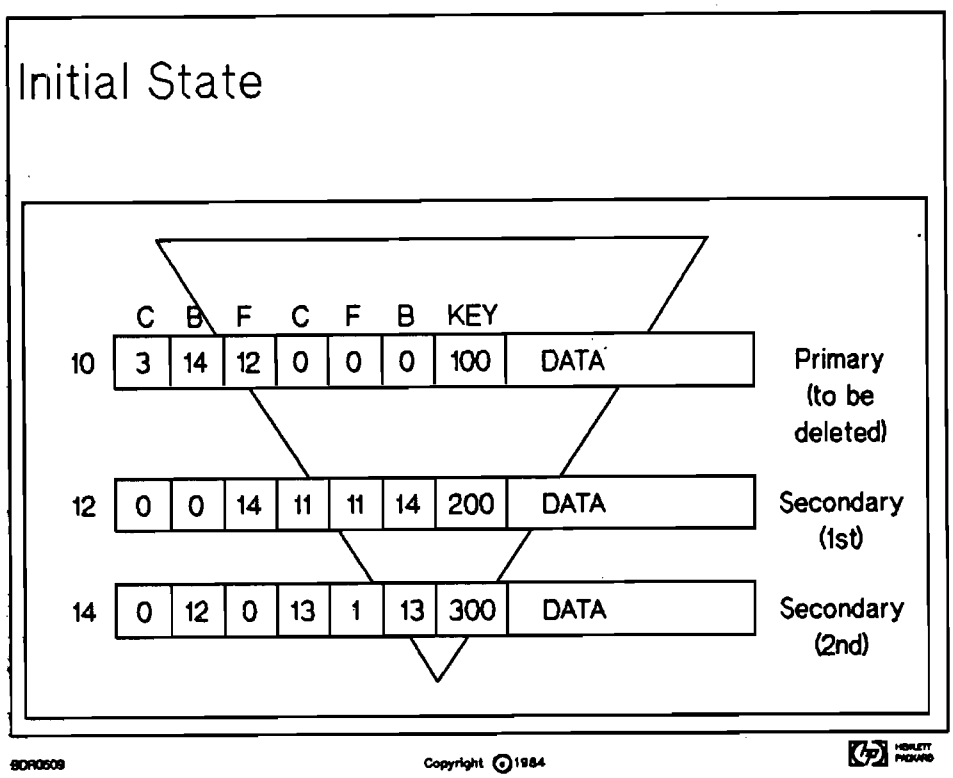
Master

1. If MASTERS-TO-BE-DELETED = 0.
   A. Post all dirty buffers.
   B. Go to "Prepare to exit".

2. Determine what type of master entry you are deleting.
   A. Call FINDRECORD on recordname in MASTERS-TO-BE-DELETED queue.
   B. Primary with no secondaries
      Go to Step 5.
   C. Primary with secondaries
      Go to Step 3.
   D. Secondary
      Go to Step 4.

□ DBDELETE

□ Notes

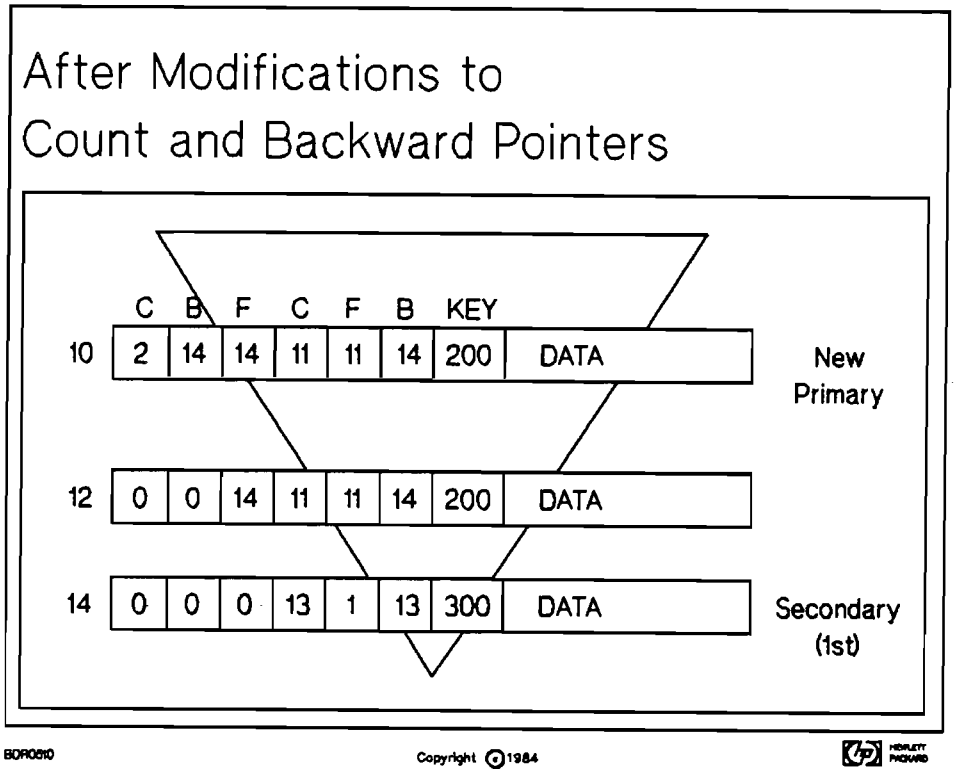


Master (cont.)

3. Deletion of a primary with secondaries.
   A. Check all count fields for zeroes.

□ DBDELETE

□ Notes



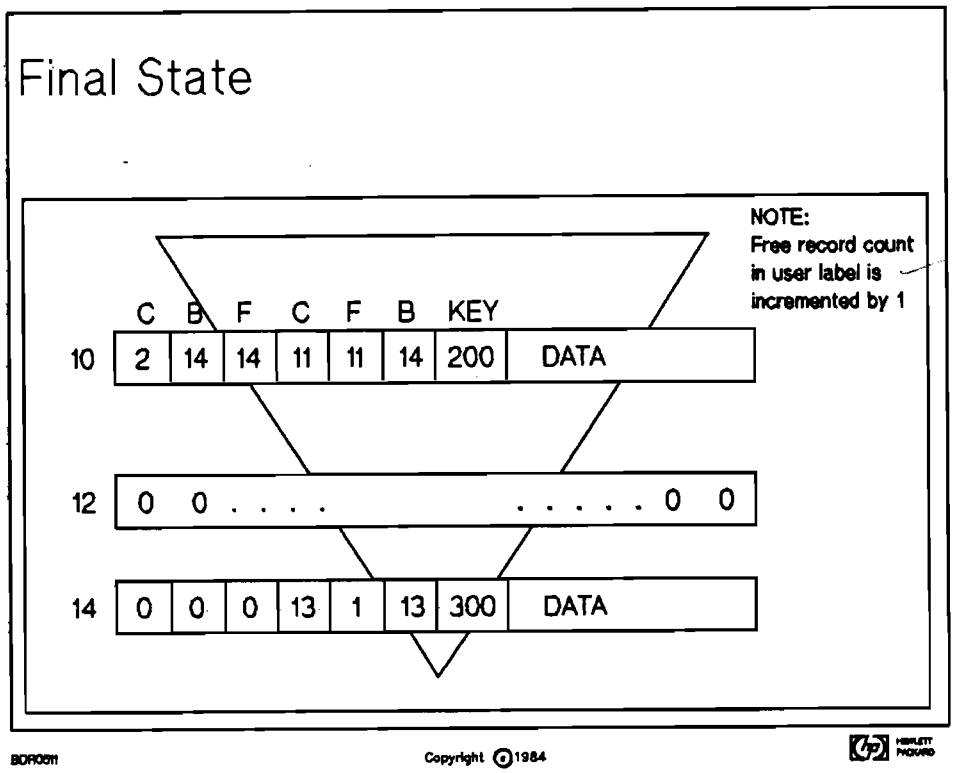
Master (cont.)

B. Call FINDRECORD on the first secondary.
   - Save the backward pointer count.
   - Move the secondary to the primary location.
   - Modify the count and backward pointer.

C. Call FINDRECORD on the next secondary.
   - Modify the backward pointer to 0.

□ DBDELETE

□ Notes

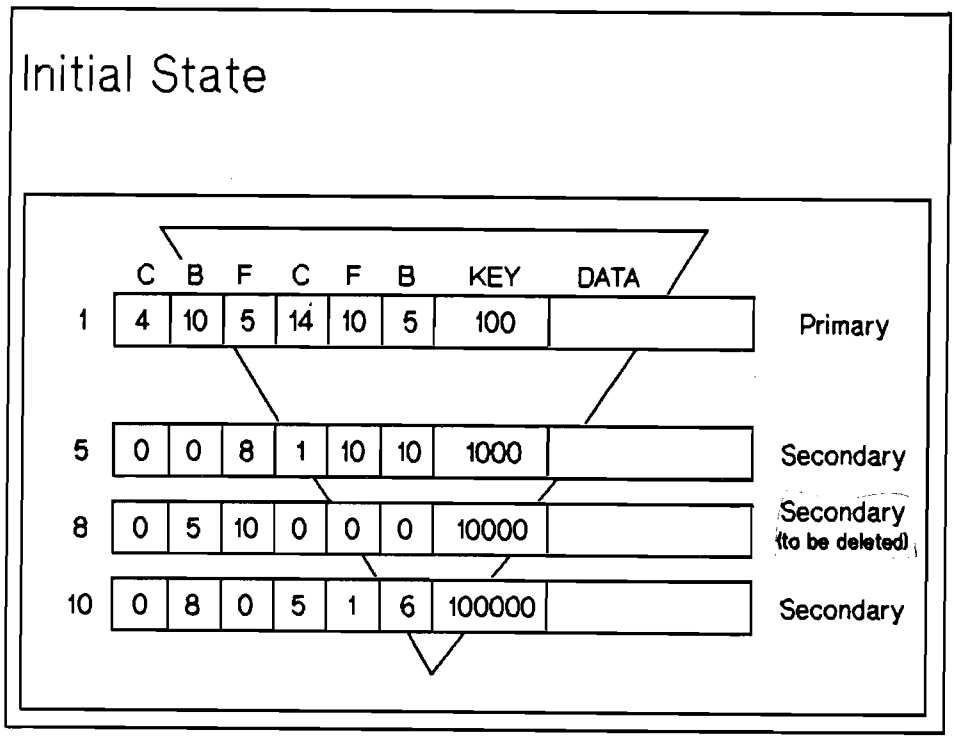


Master (cont.)

D. Set bit map and media record of first secondary to zeros.

□ DBDELETE

□ Notes

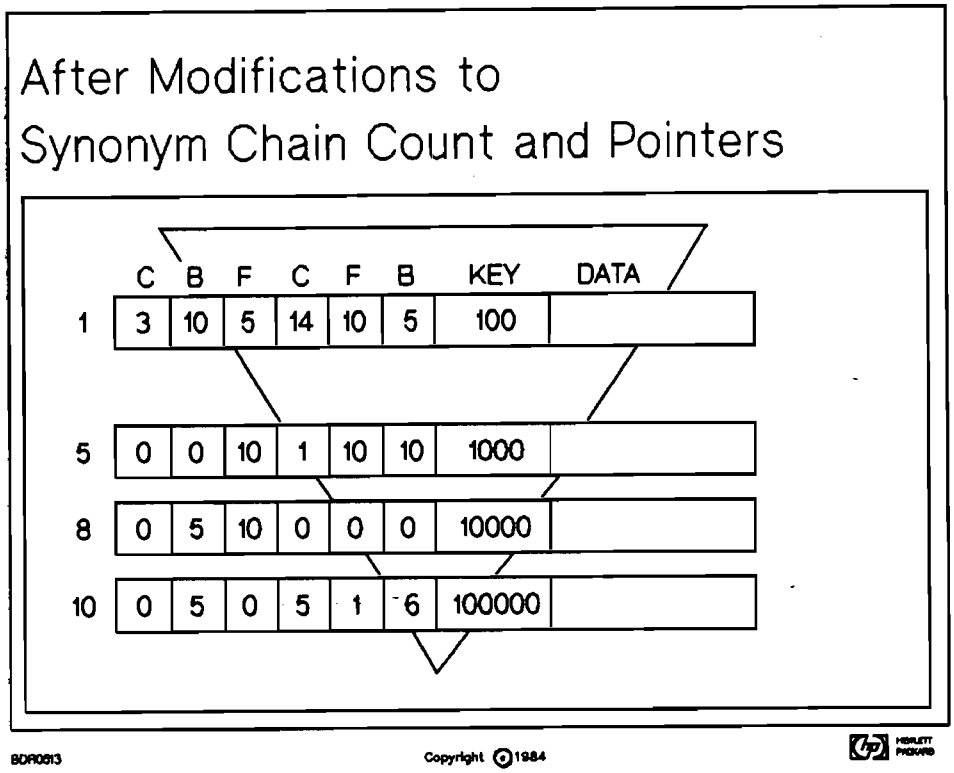


Master (cont.)

4. Deletion of a secondary.
   A. Call FINDRECORD on the forward and backward pointers.
      - Modify the forward and backward pointers of the successor and predecessor.

□ DBDELETE

□ Notes

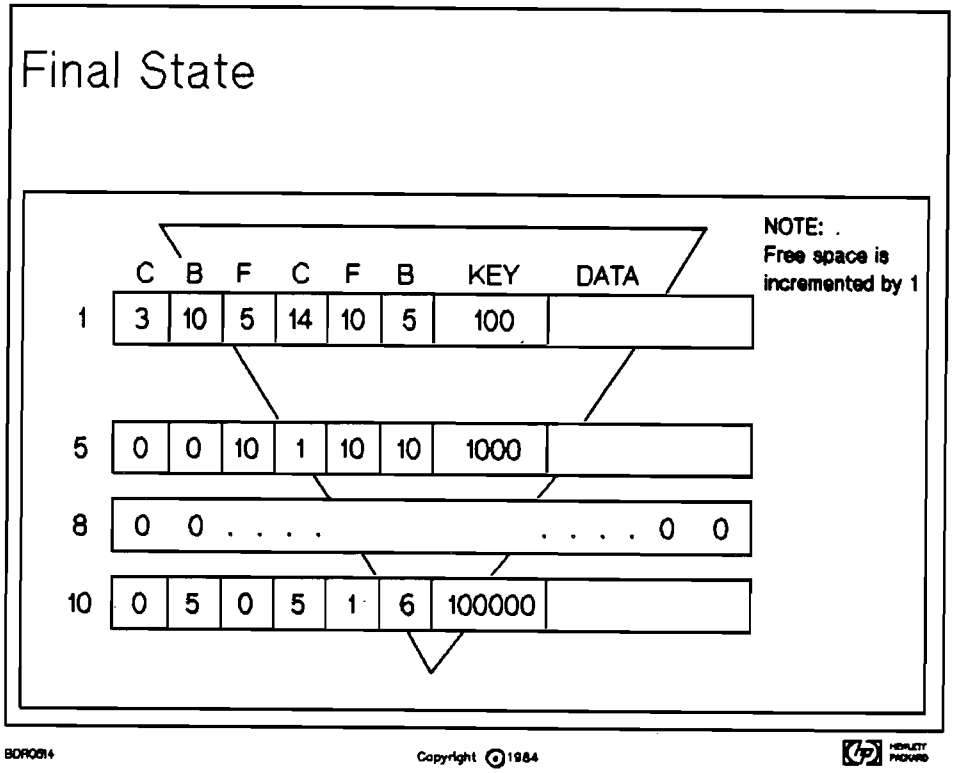


Master (cont.)

B. Hash the key value of the secondary to locate the primary (FINDRECORD).
   - If there was no predecessor to the secondary, modify the forward pointer of the primary.
   - If there was no successor, modify the backward pointer of the primary.
   - Modify the synonym chain count of the primary.

□ DBDELETE

□ Notes



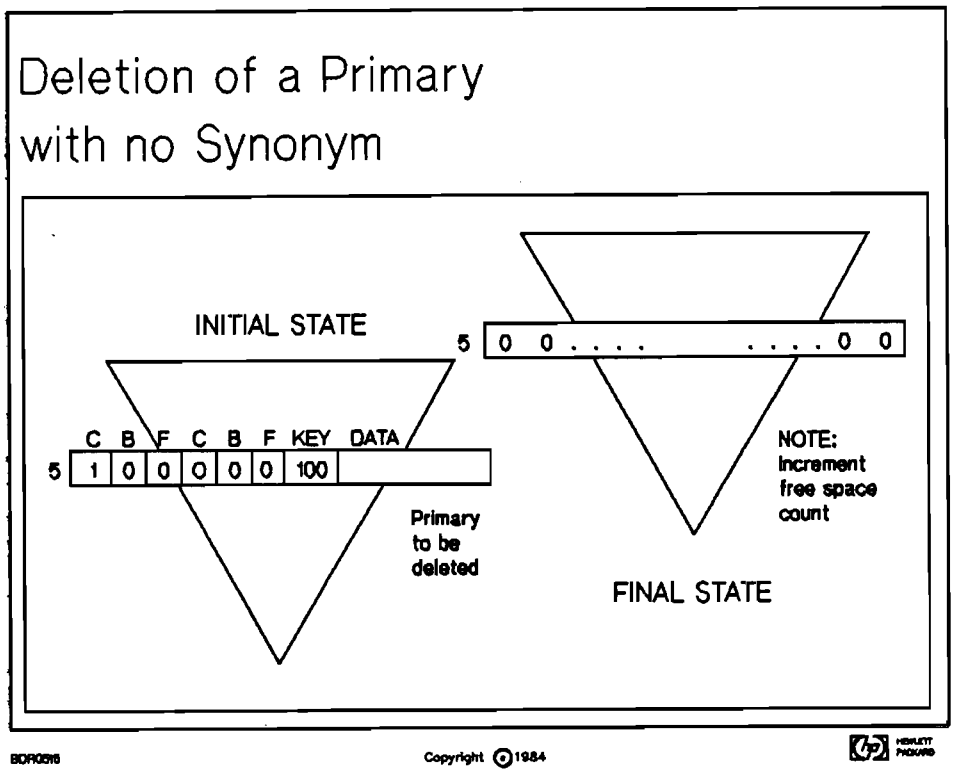
Master (cont.)

C. Set the bit map and media record of the deleted secondary to zeros.

D. Increment free space count in Set Label Table.

□ DBDELETE

□ Notes

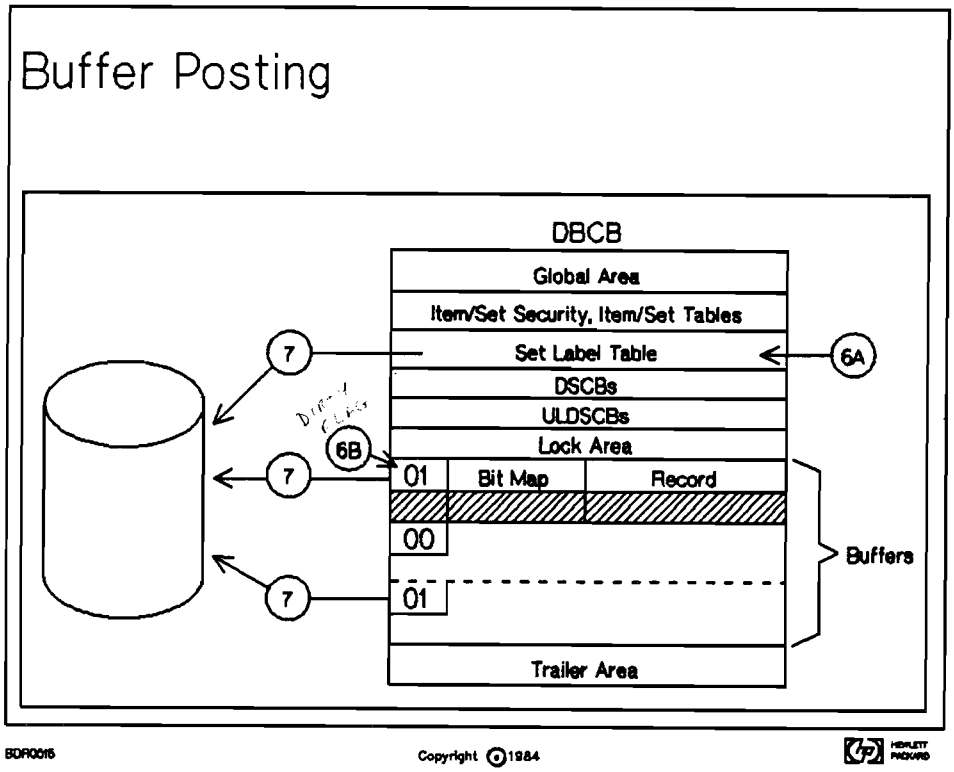


Master (cont.)

5. Deletion of a primary with no synonyms.
   A. Set the record to zeros.
   B. Set the corresponding bit to zero.

□ DBDELETE

□ Notes



Master (cont.)

6. Update user labels.
   A. Increment the free space count in Set Label Table.
   B. Mark the buffer dirty and release it.

7. Post all dirty buffers and labels to disc.

8. Prepare to exit.

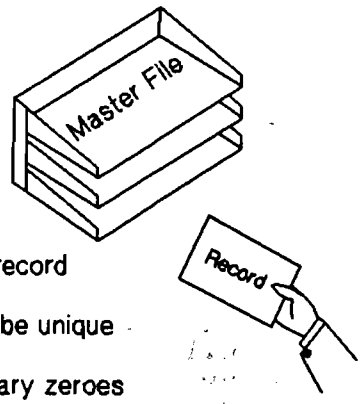
□ DBPUT

□ Notes

Adding a New Master Entry

DBPUT (Base,Dset,Mode,Status,List,Buffer)

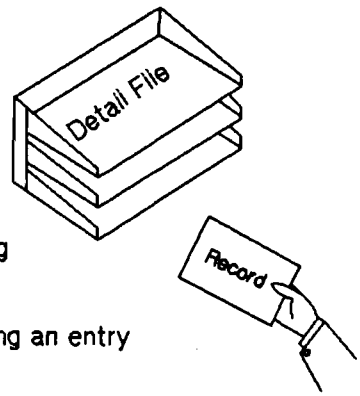
* Dset must be a manual master
* There must be space in the data set
* The search item must be in the List
* The new entry becomes the current record
* The search item value in Buffer must be unique
* Unreferenced items are filled with binary zeroes
* The search item value must be locked if the DB is opened in Mode 1



Adding a New Detail Entry

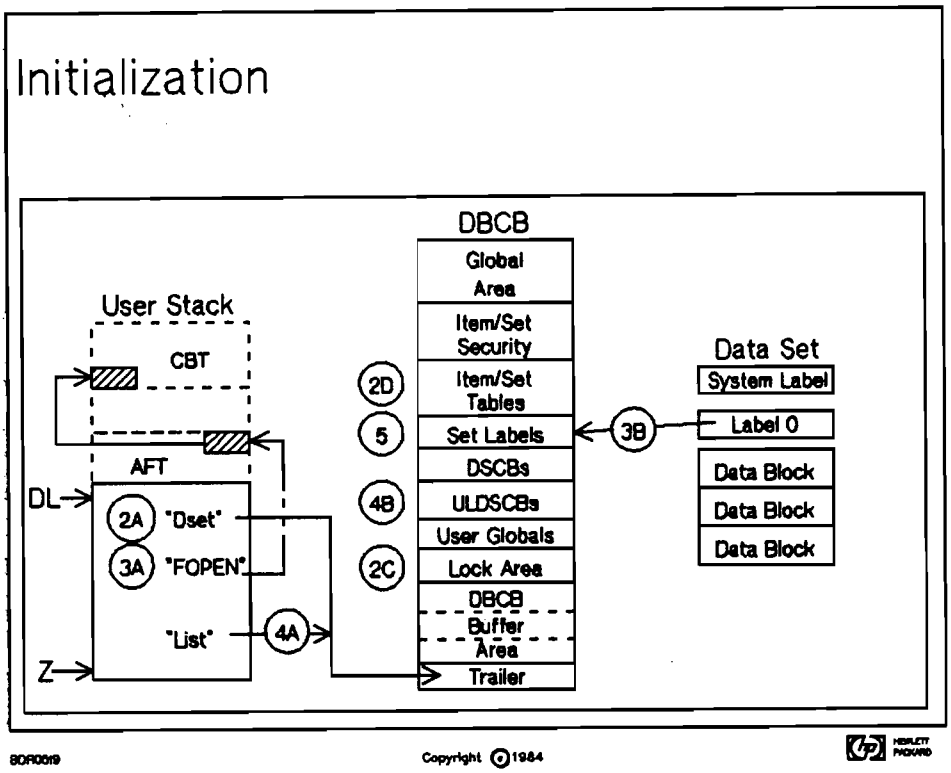
DBPUT (Base,Dset,Mode,Status,List,Buffer)

* There must be space in the data set
* All search & sort items must be in the List
* Each related manual master must contain an entry for its corresponding search item value
* If linked to an automatic not containing an entry it must have space to add the entry
* Unreferenced items are filled with binary zeroes
* The current record pointer is set to the new entry
* The data entry must be locked if the DB is opened in Mode 1



□ DBPUT

□ Notes



1. DBCB preparation

2. Verify DSET parameter and access

A. Move DSET to the trailer.
B. Modify flag must be on.
C. Check for locking if DBOPEN Mode = 1.
D. Check set type ≠ "A".

3. Prepare for data set access

A. Call FOPEN if DSET is not opened.
B. Issue FREADLABEL on data set Label 0.

4. Validate the list parameter

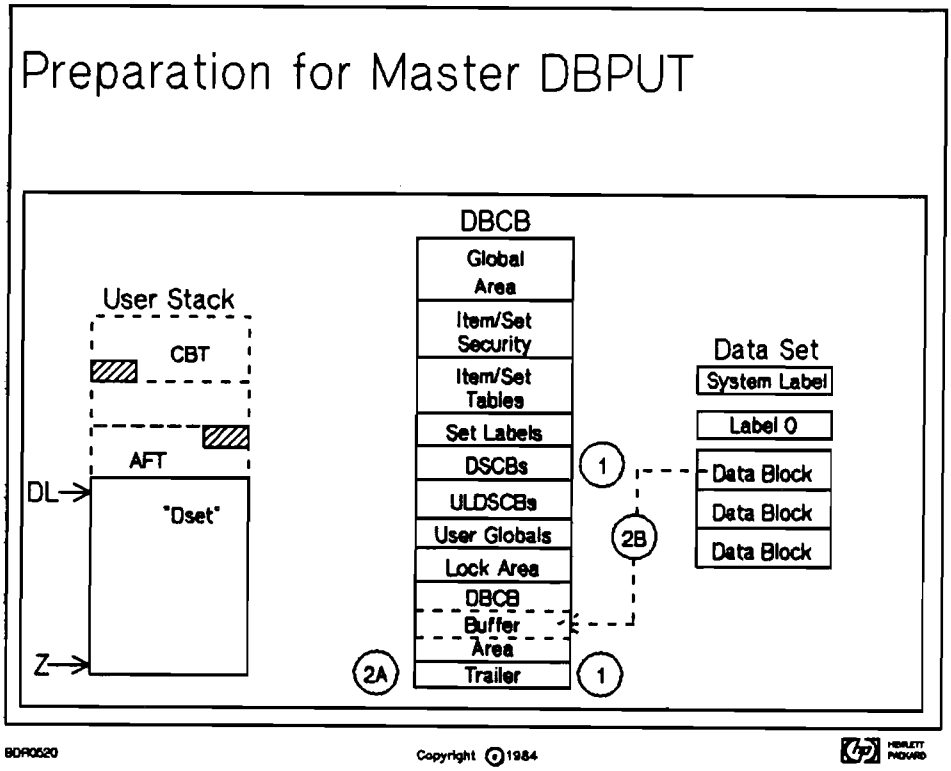
A. Move the list to the trailer.
B. Set the "current list" for this data set.
C. List processing will fail if an item is:
   - Duplicated, not writeable, not in the set.

5. Check Label 0 to insure data set capacity is not reached

6. If type = "M" perform "master" If type = "D" perform "detail"

□ DBPUT

□ Notes



Master

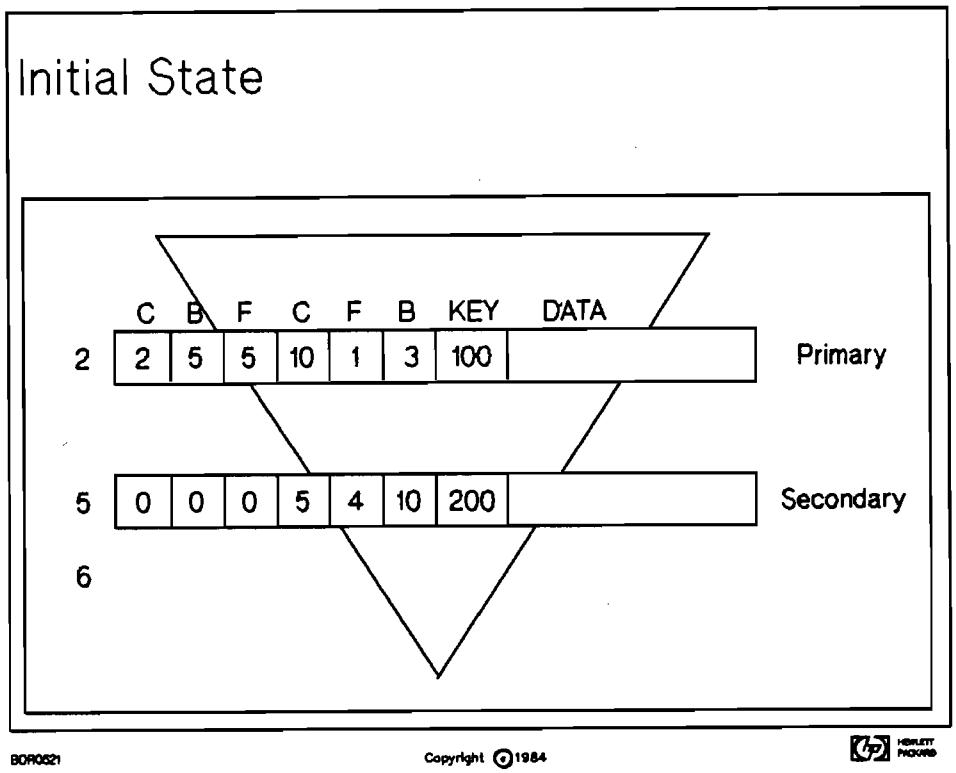
1. Verify the key item is in the current list.

2. Find the primary location using the key value.
   A. Hash the key.
   B. Call FINDRECORD.

3. Determine if the primary location is occupied.
   A. Occupied by a synonym.
      - Go to Step 4 (Case A).
   B. Occupied by a primary.
      - Go to Step 3 (Case B).
   C. If not occupied (bit = 0).
      - Go to Step 5 (insert the record).

□ DBPUT

□ Notes

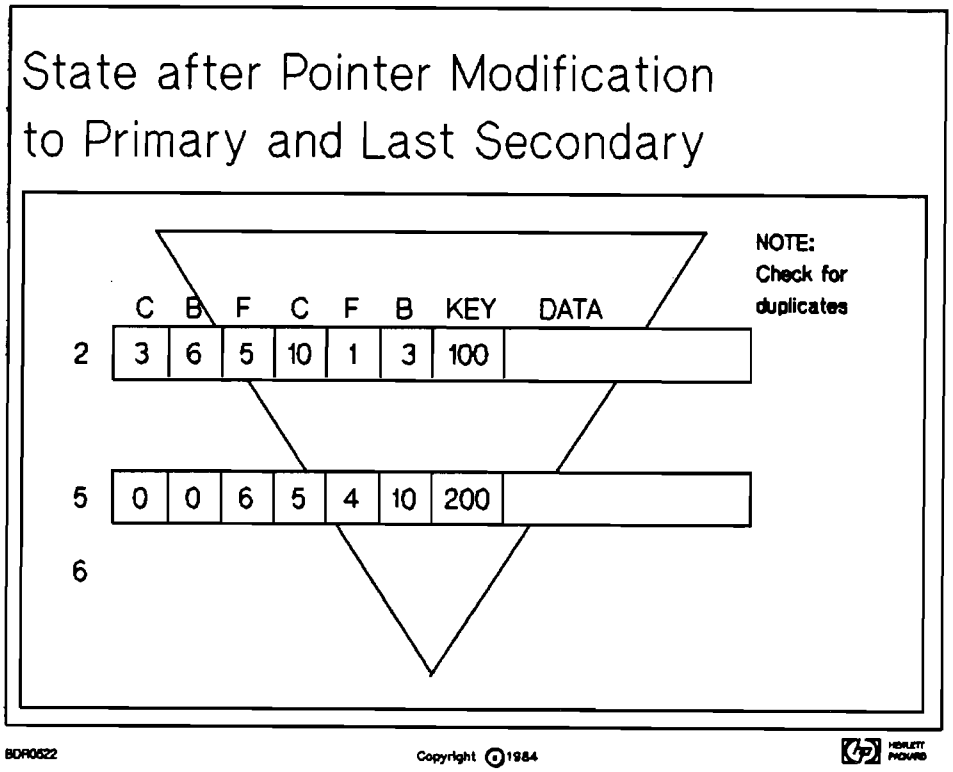


Master (synonym chase)

4. The primary location is occupied by another primary.
   A. Call FINDRECORD on each synonym to check for duplicates.
   B. Locate the end of the synonym chain.
      - Search the bit map for the nearest empty location (may require a call to FINDRECORD).

□ DBPUT

□ Notes

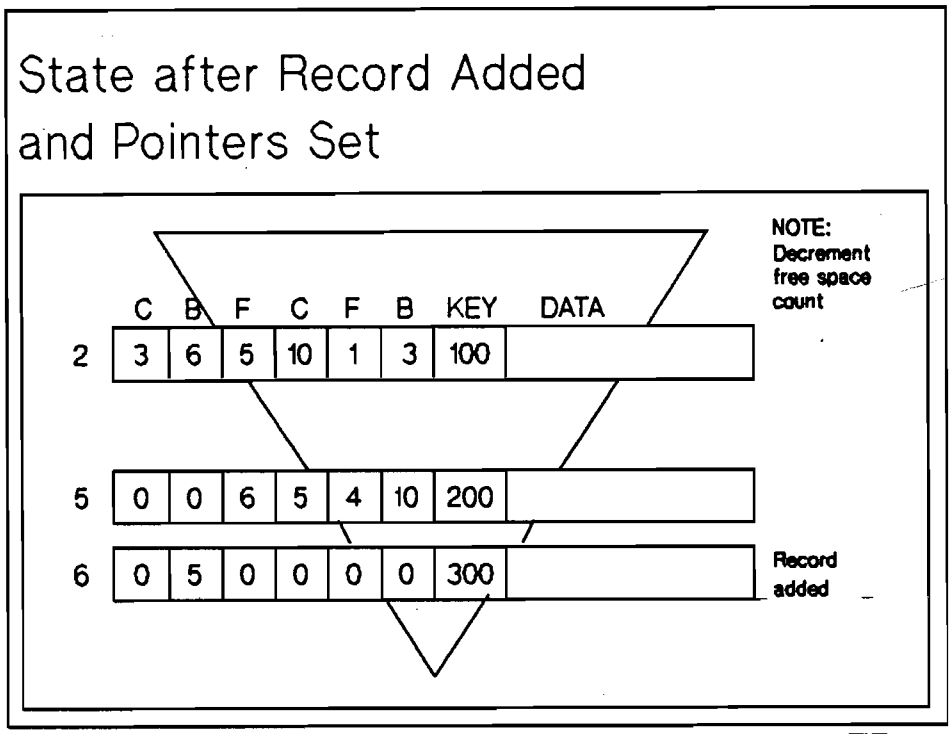


Master (synonym chase – cont.)

C. Update the forward pointer of the last synonym.
D. Update the backward pointer and chain counts of the primary (FINDRECORD).

□ DBPUT

□ Notes

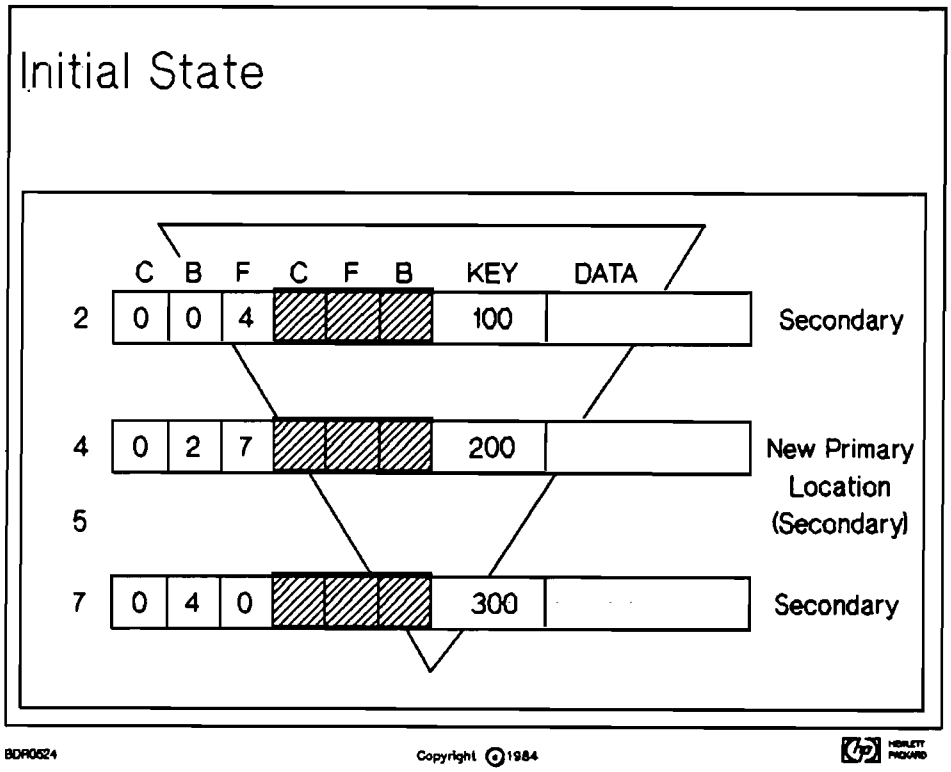


Master (synonym chase – cont.)

E. Add the record.
   - Set bit map to 1.
   - Set the forward and backward pointers.
   - Make the record the current record (ULDCB).

□ DBPUT

□ Notes

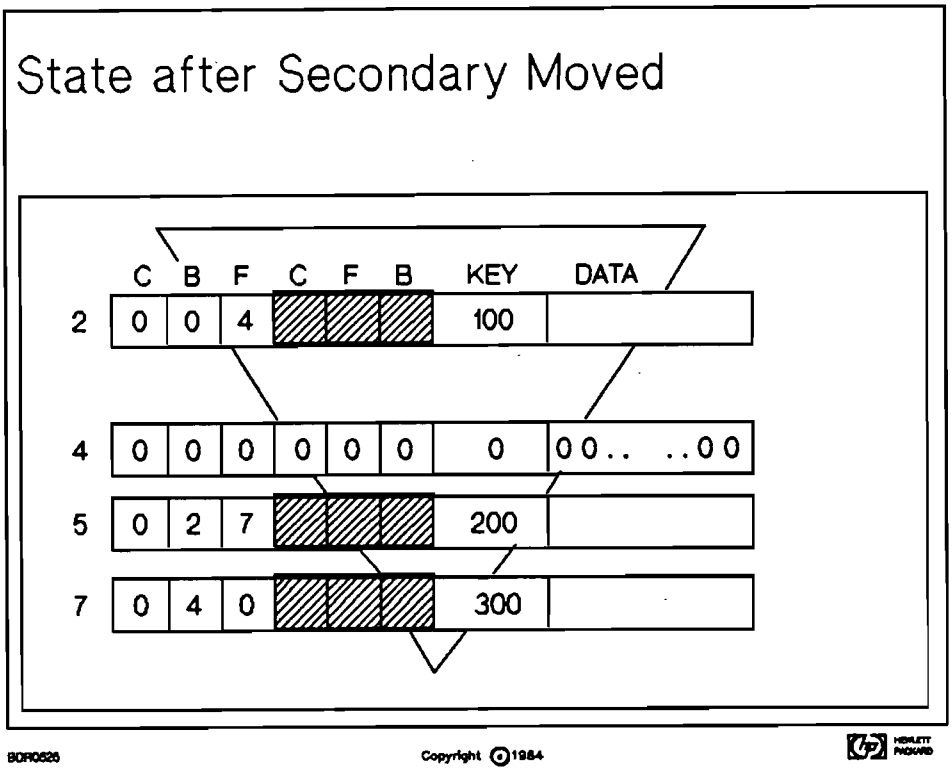


Master (migrating secondary)

5. Primary location is occupied by a synonym.
   A. Search for first free location in the data set.

□ DBPUT

□ Notes

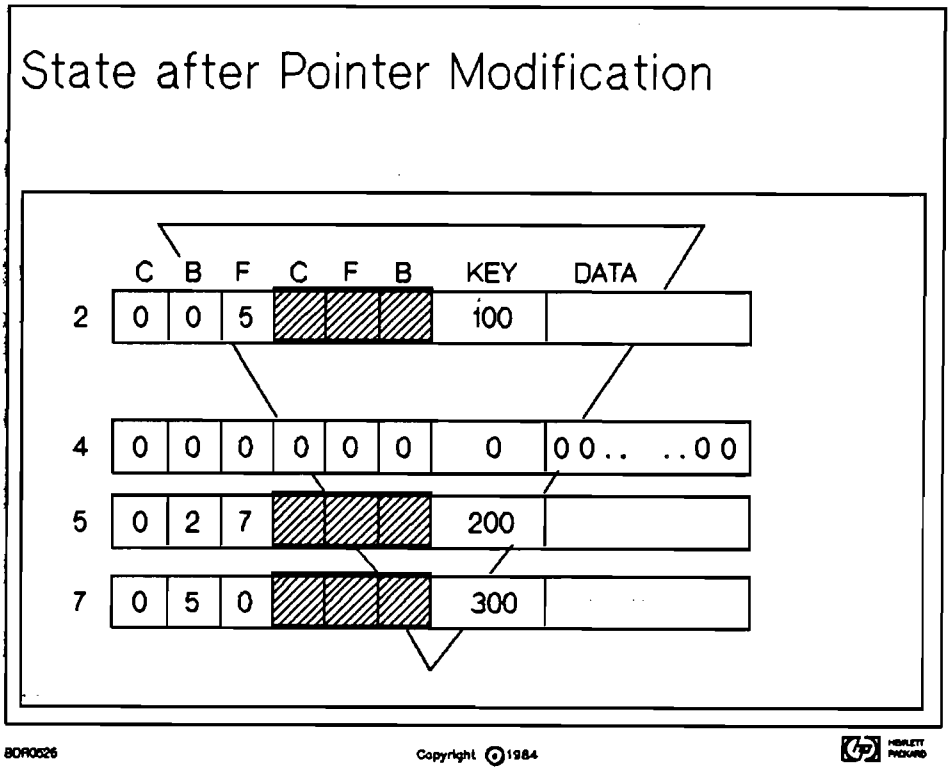


Master (migrating secondary – cont.)

B. Move the synonym to the free space.
C. Set the media record in the old location to zeros.

□ DBPUT

□ Notes

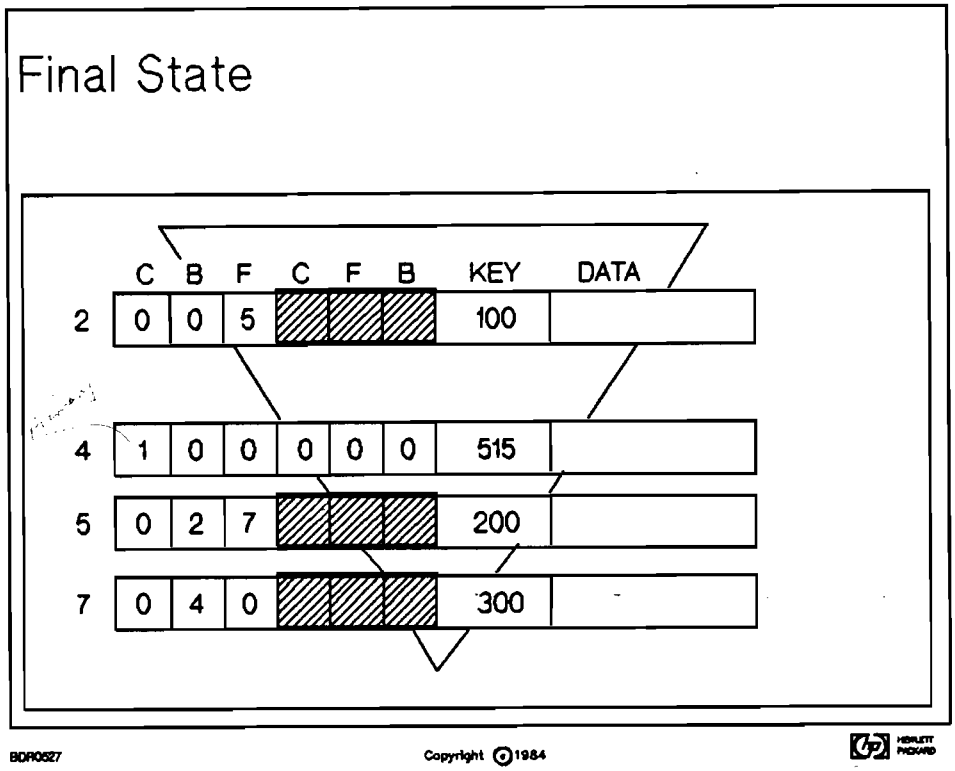


Master (migrating secondary – cont.)

D. Call FINDRECORD on the forward and backward pointers of the secondary.
   - Modify the forward pointer of the synonym predecessor.
   - Modify the backward pointer of the synonym successor.

□ DBPUT

□ Notes

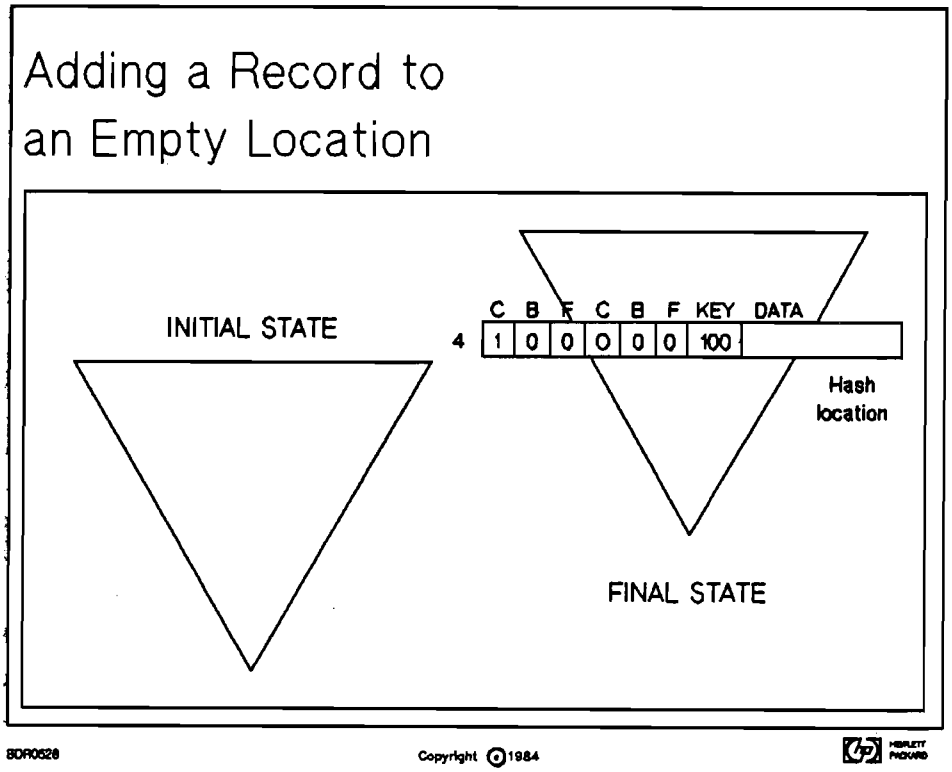


Master (migrating secondary – cont.)

E. Insert the record (FWRITEDIR)
   - Set bit map = 1.
   - Set synonym pointers to zero.
   - Decrement free space count in Label Table.

☐ DBPUT

☐ Notes

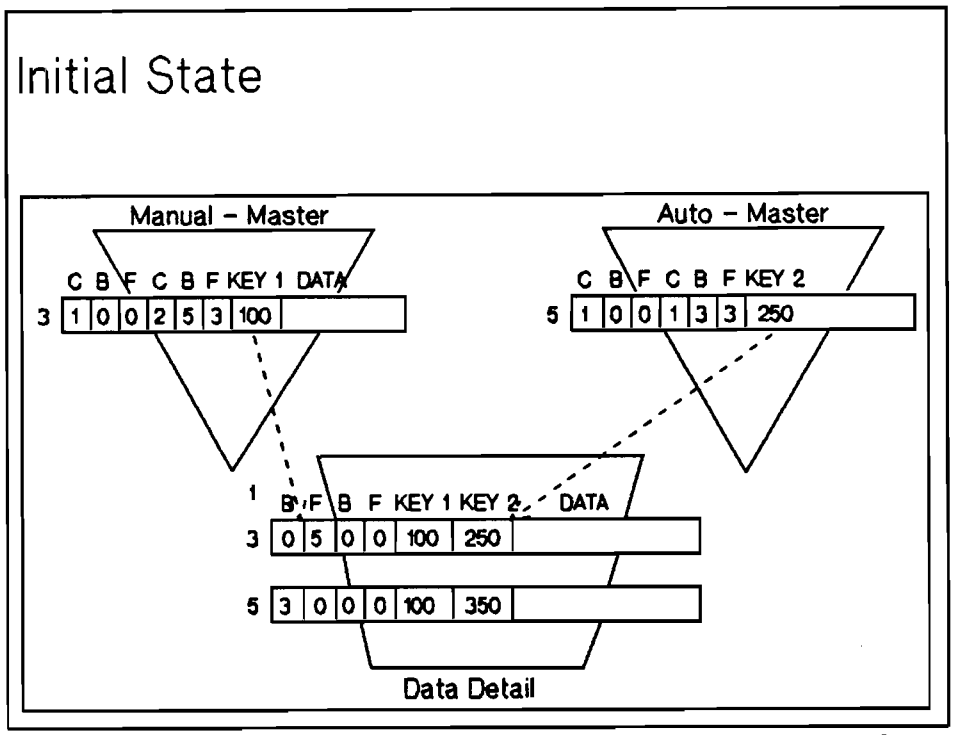


Master (empty location)

6. Copy the record to the appropriate location.
   A. Set the bit map to 1.
   B. Set the forward and backward synonym pointers to 0.
   C. Set count to 1.
   D. Decrement free space count (Label Table).

□ DBPUT

□ Notes

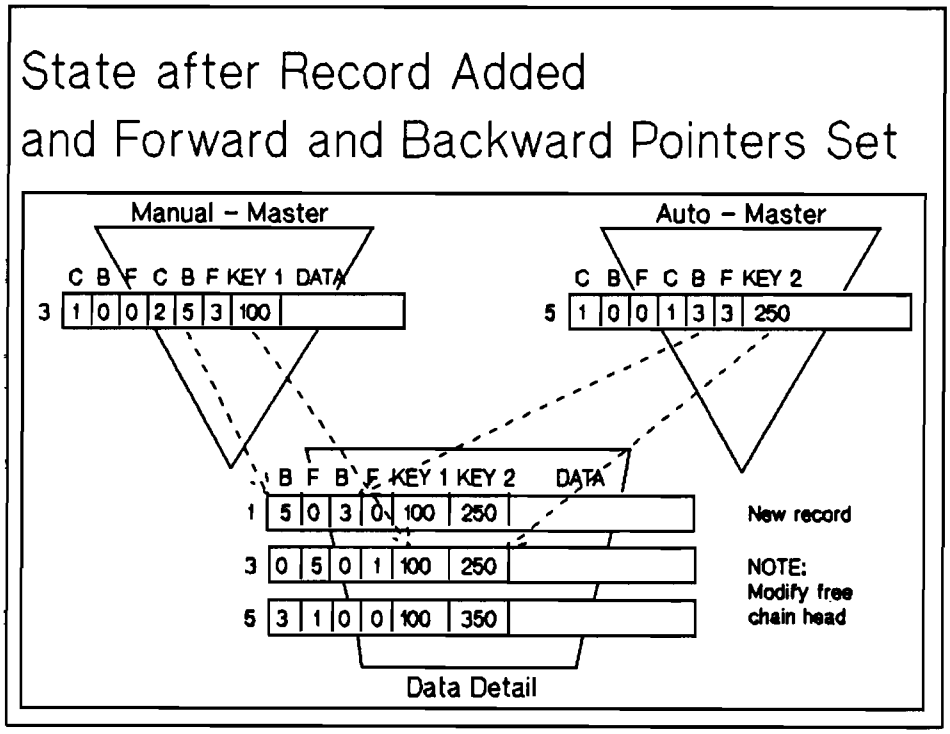


Detail

1. Adding a detail record.
   A. Check free chain head (Label 0) for a reusable detail entry or use the data set EOF.
      – Call FINDRECORD to locate the block.

□ DBPUT

□ Notes

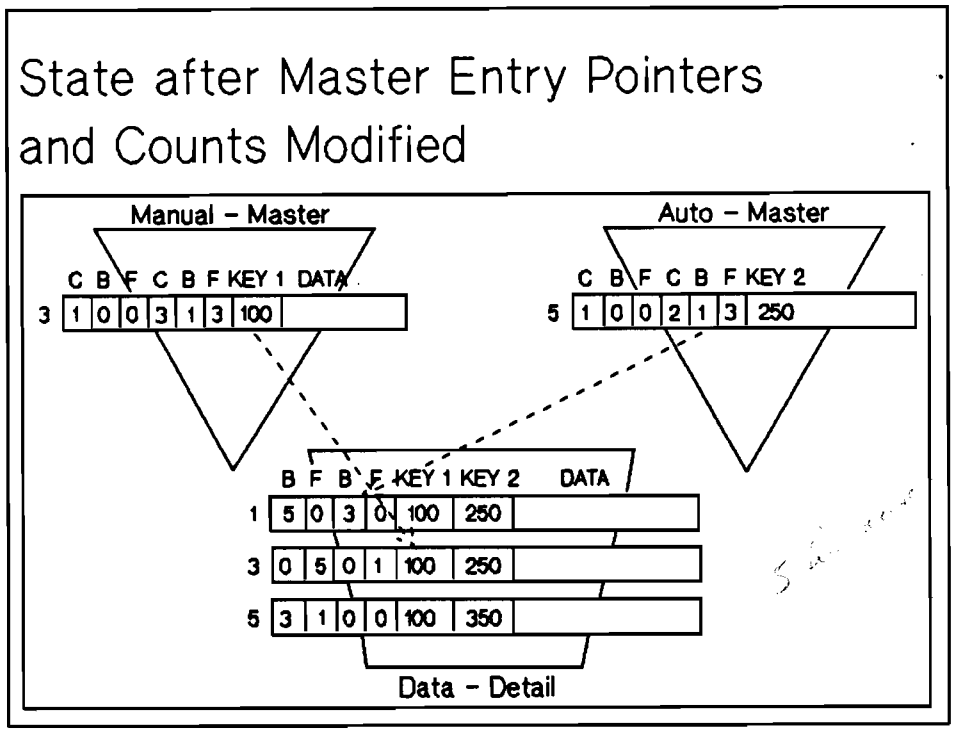


Detail (cont.)

B. Insert the record.
   - Modify pointers in the detail chain.
   - Set the bit to 1.

□ DBPUT

□ Notes

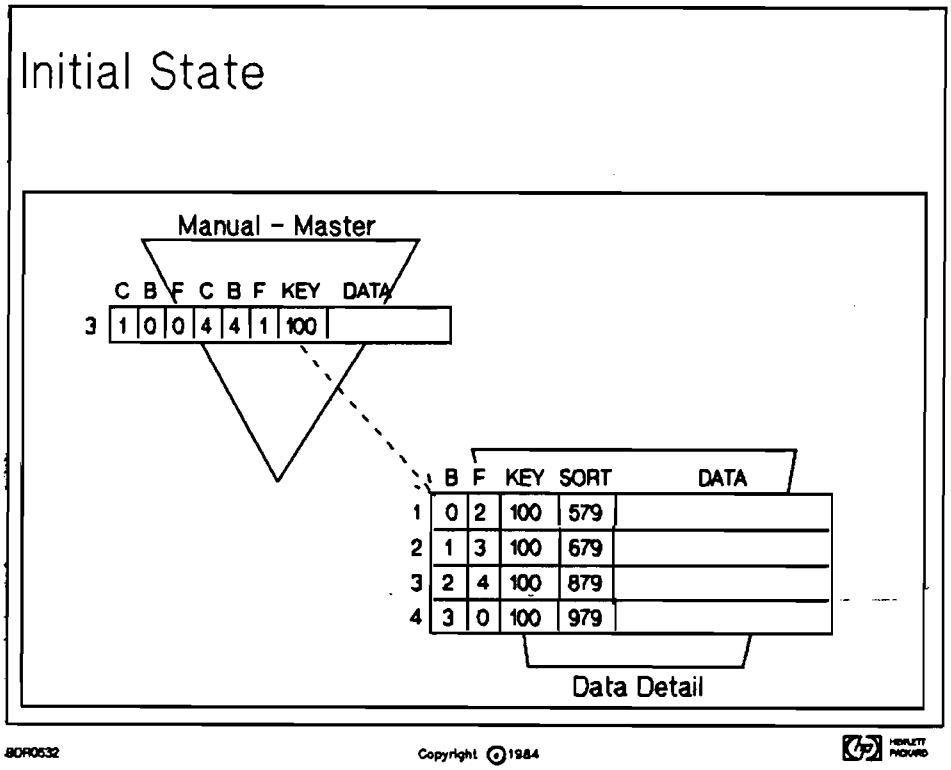


Detail (cont.)

C. For each path (hold detail to use search item to locate masters - hash):
   - Modify the backward pointer and chain count of master (FINDRECORD).
   - Modify previous record forward pointer (FINDRECORD).
   - Set the forward and backward pointers of the new entry.

□ DBPUT

□ Notes



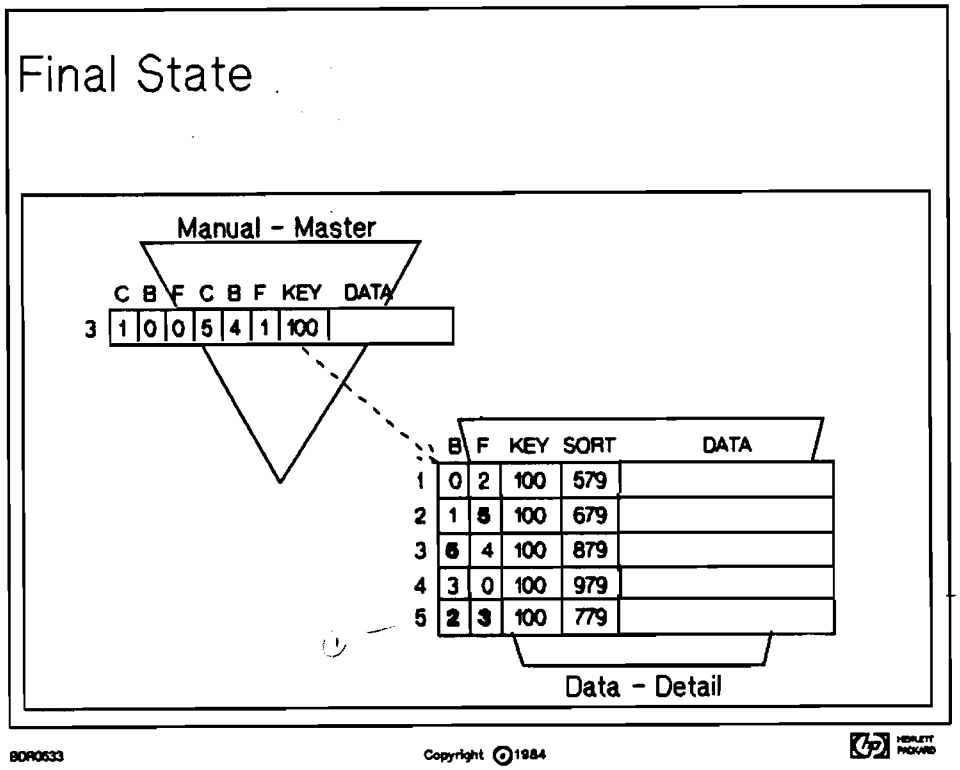
Sorted Chains

1. Check the free chain head and EOF for space to insert new record.
   - Insert the record (set bit map to 0).

2. Call FINDRECORD on the master entry.
   A. Locate the last record on the chain.
   B. Call FINDRECORD on the backward pointer.

□ DBPUT

□ Notes



Sorted Chains (cont.)

3. Check the sort field on the last read.
   A. If new sort value = old sort value:
      - Extend the sort field to the rest of the record.
   B. If new sort value > old sort value:
      - Set forward and backward pointers of new entry.
      - Modify master record chain count.
      - Modify forward pointer of preceding record on chain.
      - Modify backward pointer of succeeding record on chain.

Module 5.37

Worksession

1. For each of the procedure calls determine the minimum number of data blocks or labels which would need to be read if the data bata was optimally designed for that particular procedure.

2. Next, identify the situations which could exist causing more than this minimum to be read.

3. Finally, suggest some guidelines to minimize the overheads you mention in Step 2.

Module 5.38

	Minimum # of calls to FINDRECORD (blocks and labels).	Minimum # of disc WRITES.	Could there be more FINDRECORDS & WRITES to disc.	Design guidelines to minimize # of disc I/O's.
DBUPDATE	**Data Set:**	**Data Set:**	**Data Set:**	**Data Set:**
	1	1	No	more buffers
DBDELETE (master)	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**
1. Primary with no secondaries	1	2 — buffe	1.e	more buffers
2. Primary with secondaries	1 ×	2 — buffe	If more	x large
3. Secondary	1 × (primary)	2 — label		x large buffers
	+ 1 master record		4 is	
	Detail Data Set:	**Detail Data Set:**	**Detail Data Set:**	**Detail Data Set:**
	0			

Module 5.39

	Minimum # of calls to FINDRECORD (blocks and labels).	Minimum # of disc WRITES.	Could there be more FINDRECORDS & WRITES to disc.	Design guidelines to minimize # of disc I/O's.
DBDELETE (detail)	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**
	Detail Data Set:	**Detail Data Set:**	**Detail Data Set:**	**Detail Data Set:**

Module 5.40

	Minimum # of calls to FINDRECORD (blocks and labels).	Minimum # of disc WRITES.	Could there be more FINDRECORDS & WRITES to disc.	Design guidelines to minimize # of disc I/O's.
DBPUT (master)	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**
1. Hash to a secondary 2. Hashes to primary with secondaries	**Detail Data Set:**	**Detail Data Set:**	**Detail Data Set:**	**Detail Data Set:**
DBPUT (detail)	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**	**Master Data Set:**
	Detail Data Set:	**Detail Data Set:**	**Detail Data Set:**	**Detail Data Set:**

Shared Access Considerations

Module 6.2

Module Outline

Shared Access

0. Introduction
1. Locking Levels
2. DBLOCK

Lab

3. DBUNLOCK
4. Strategy and Guidelines
5. MR Capability

Quiz

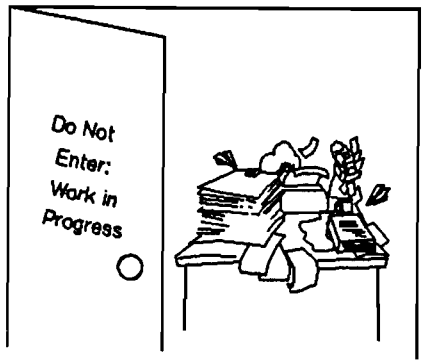
□ Introduction

□ Notes

Locking

* A solution to data inconsistency

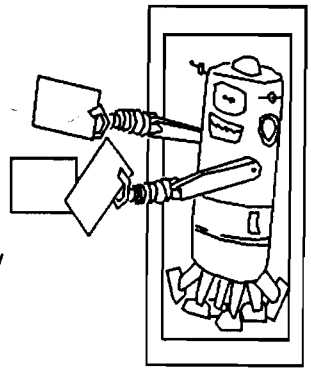
* Allows users to gain exclusive control to a portion of the IMAGE Data Base



The Transaction

* Locking is related to a total transaction consisting of several calls to IMAGE intrinsics to locate and modify the data

* At the start of any transaction establish locks that cover all data entries you intend to modify and all data entries which must not change during the transaction



□ Introduction

□ Notes

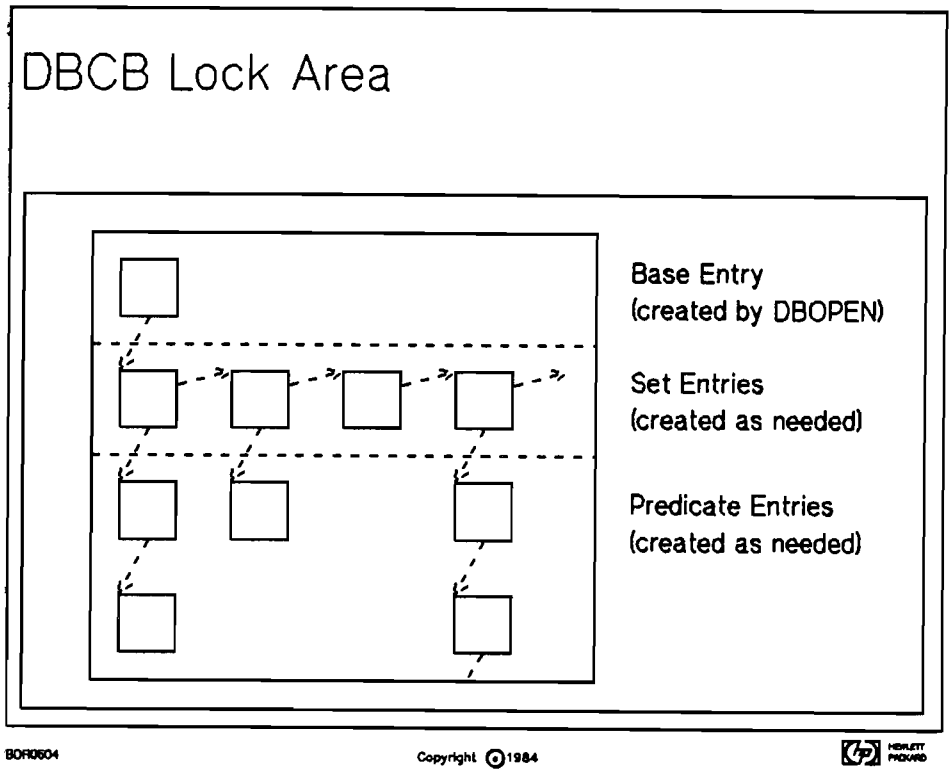
Locking - How Does it Work?

1. A process issues a lock request
2. IMAGE checks the lock area for a particular lock entry
   * If no conflict exists - IMAGE will either lock the existing entry or add a new one to the lock area
   * If a conflict exists:

For Unconditional Locking:	The process suspends until the conflicting entities have been unlocked
For Conditional Locking:	Process is not suspended Image returns control immediately with exception code

☐ Locking Levels

☐ Notes



1. Base Level
   A. Created by DBOPEN.
   B. Always exist in lock area of DBCB.
   C. Must gain control of this level before processing to lower levls.

2. Set Level
   A. Created as needed by DBLOCK.
   B. One set level for each which has a lock applied to it.
   C. Must gain control of this level before processing to the item level.

3. Item (or Predicate) Level
   A. Created as needed.
   B. One item level for each lock item within a data set.

NOTE: To gain control of a set level, it is not necessary to gain control of all set levels.

□ Locking Levels

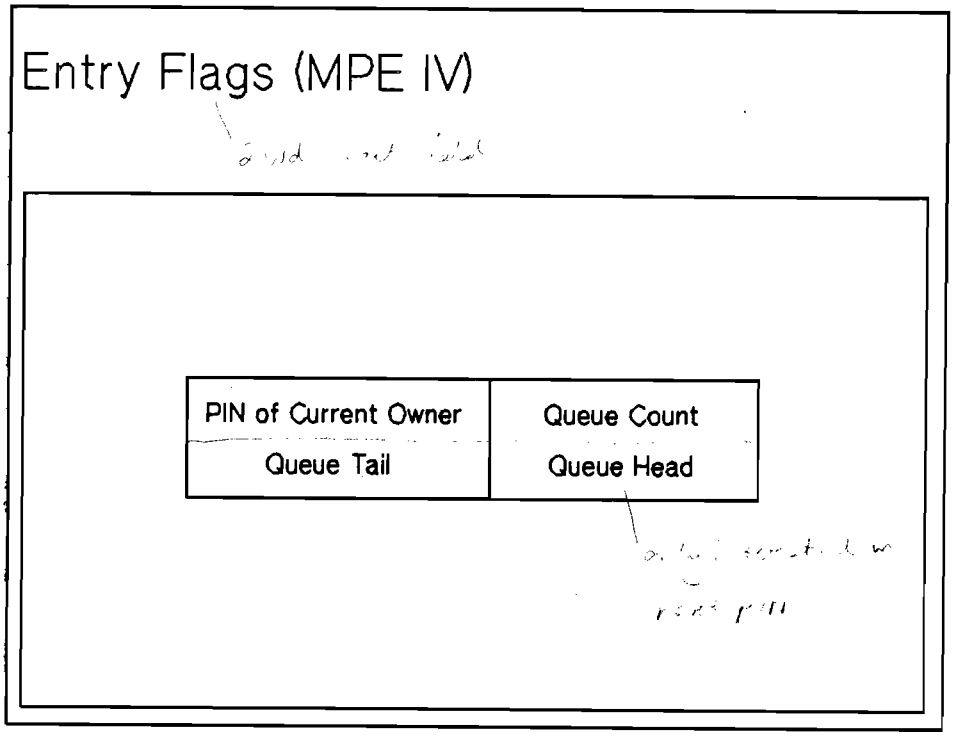
□ Notes

Locking Levels

* IMAGE allows three levels of locking:
    1) An entire data base
    2) Entire data sets
    3) A set of data entries
* IMAGE allows various users to lock at different levels concurrently which may cause conflict
    - If a data base lock is requested it can not succeed until all dataentries and data set locks are released
    - If a set lock is requested it can not succeed until all data entries are released
    - All lock requests are queued as a function of time

☐ Locking Levels

☐ Notes

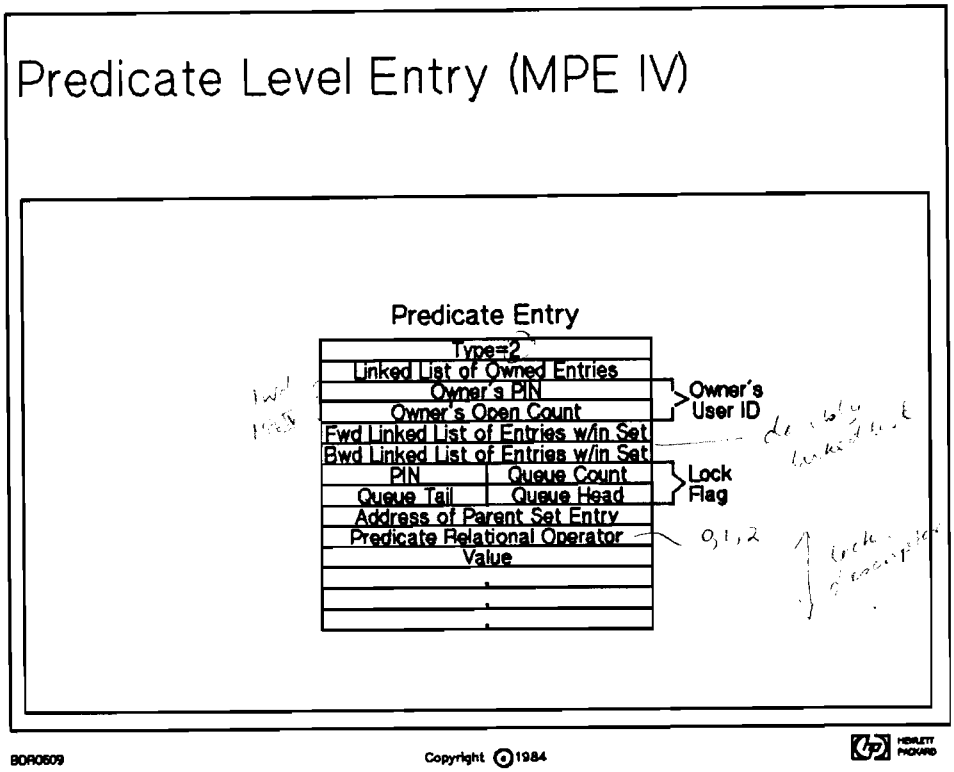


Two types of flags.

1. Lock flag: used to gain control of the entry.
2. Busy flag: used to gain control of all levels below.

□ Locking Levels

□ Notes

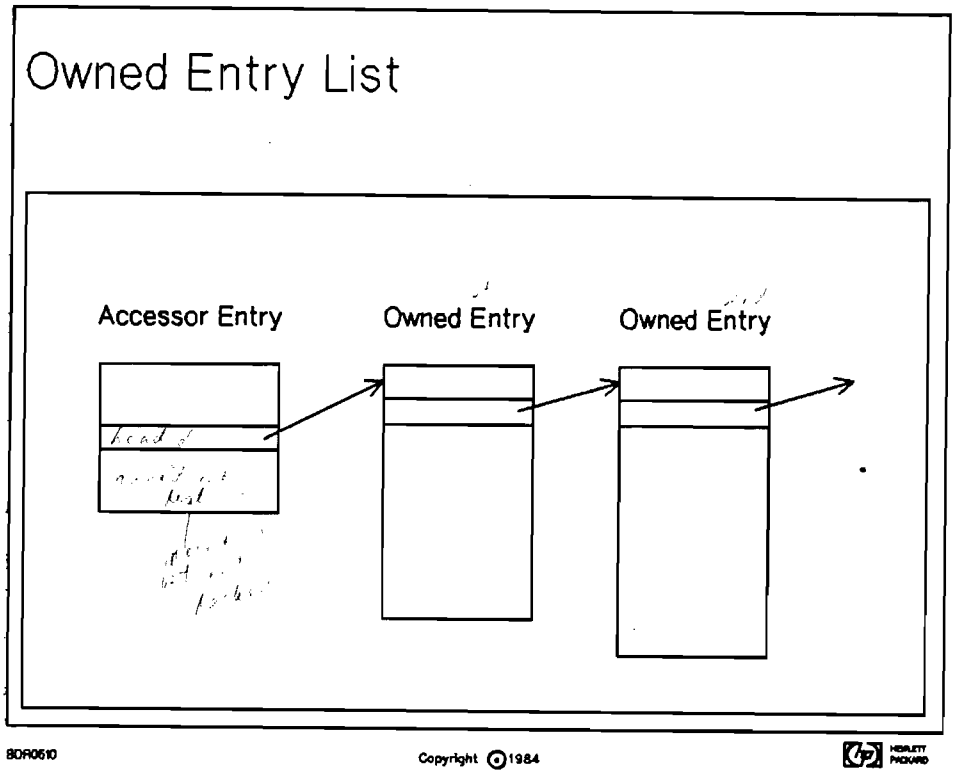


Lock flag used as before.

No busy flag as no lower levels.

☐ Locking Levels

☐ Notes



Head is in accessor entry.

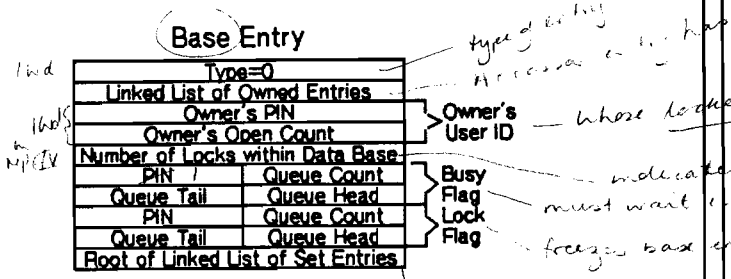
Each owned entry points to next in list.

Module 6.8

□ Locking Levels

□ Notes

Base Level Entry (MPE IV)



BDR0607

Type specifies type of level entry (base is type 0).

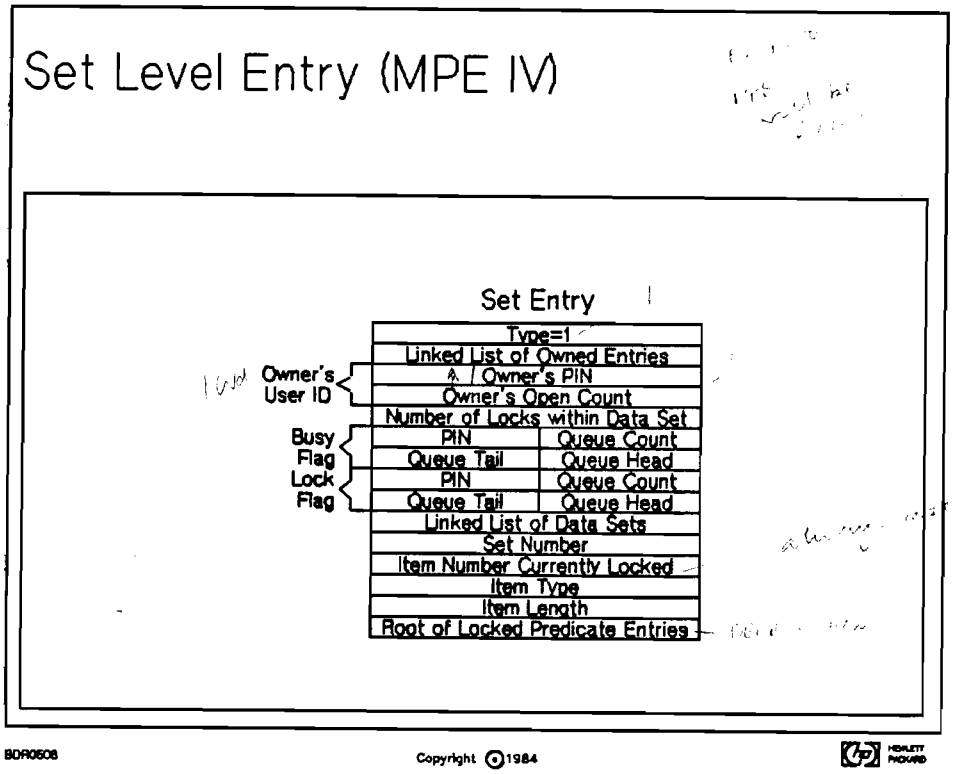
Owner's user ID in pin number and open count.

Busy flag is used if there are locks within the Data Base (number of locks within Data Base > 0). Pin of current owner in busy flag
- is 0 if no locks within.
- is 1 if > 0 locks within.

Lock flag is used to gain control of base level entry (all users trying to gain control are queued here).

□ Locking Levels

□ Notes



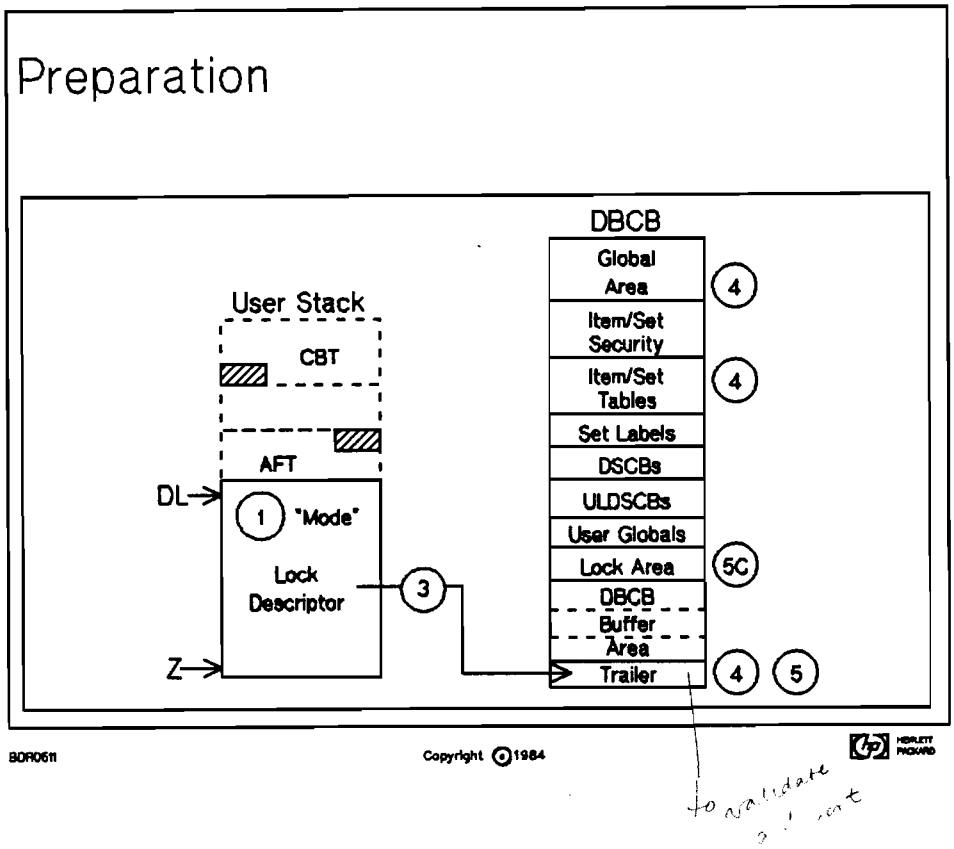
Set number refers to set currently locked by this entry.

Item currently locked defines the lock item.

Lock flag and busy flag used as in base level entry.

□ DBLOCK

□ Notes



1. Parameter validation.
   - Validate mode (1-6).

2. DBCB preparation.
   - No information is moved from ULCB to DBCB.

3. Move the lock descriptor to the trailer.

4. Validate and translate the lock descriptor.
   A. Transform set name to a set number.
   B. Transform item name to an item number.
   C. Verify the item is not compound.
   D. Verify the item length.

5. Sort the lock descriptors in acsending order.
   A. Sort sets by set number.
   B. Sort descriptors in the same set by lower bound.
   C. Set up the accessor entry.

6. For each lock descriptor, process a lock to the level required.
   - If no more descriptors, go to 10.

□ DBLOCK

□ Notes

Base Level Lock

Conditional vs. Unconditional

Request	Number of Locks	Action
Conditional (Mode=2)	=0	Set Owner's User ID equal to Open Count & PIN
Conditional (Mode=2)	>0	Unfreeze the Base Level Entry & return to the user
Unconditional (Mode=1)	=0	Set Owner's User ID
Unconditional (Mode=1)	>0	Wait (Busy Flag)

BDR0612

7. Data base level.
   A. Freeze base level entry (lock flag).
      - A conditional request returns if entry is locked.
      - An unconditional request waits if entry is locked.
   B. Check the DBLOCK mode.
      - If Mode = 1 or 2 (data base lock), check number of locks.
      - If Mode ≠ 1 or 2, increment number of locks unfreeze base level entry. Go to set level entry.

□ DBLOCK

□ Notes

Set Level Lock

Conditional vs. Unconditional

Request	Number of Locks	Action
Conditional (Mode=4)	=0	Set Owner's User ID
Conditional (Mode=4)	>0	Unfreeze the ~~Base~~ SET Level Entry, decrement # of locks w/in DB, & return to user
Unconditional (Mode=3)	=0	Set Owner's User ID
Unconditional (Mode=3)	>0	Wait (Busy Flag)

8. Data set level.
   A. Create set level entry if needed (may expand the lock area).
   B. Freeze the set level entry (lock flag).
      - Conditional request returns to user if entry is frozen.
      - Unconditional request waits if entry is frozen.
   C. Check mode parameter, if = 3 or 4 (data set lock), check number of locks in data set.
   D. If Mode ≠ 3 or 4,
      - Increment number of locks in data set.
      - Unfreeze set level entry.
      - Go to item level.

□ DBLOCK

□ Notes

Record Level Lock

Conditional vs. Unconditional

Request	Number of Locks	Lock Item	Action
Conditional (Mode=6)	>0	or Not Current	Unfreeze Set Level Entry, & return to user
Unconditional (mode=5)	>0	or Not Current	Wait, set Item number currently locked in set entry, increment # of locks in set, & unfreeze set entry
Conditional or Unconditional	=0	and Current	Go to 9(C)

9. Item set entry.
   A. Check mode parameter, if = 5 or 6.
   B. Scan for conflicting item values or conflicting lock items. If found and . . .
      - Conditional request: decrement number of locks for all levels and return to user.
      - Unconditional request: wait.
   C. Create item level entry if needed. If successful, return to user.

10. Prepare to exit.

Module 6.16

Lab

Using DBDRIVER.PUB.SYS,PRIV do the following:

1. Open the EXAMPL data base in mode 1.
2. Perform a predicate lock on an ~~account~~ (Name) item - ~~detail~~ (master). (Account master)
3. Use DEBUG to find the predicate entry associated with your lock.

NOTE: Ask your instructor for the location within the DBCB of the base of the PIN Hash Root Table. %201

/PRED

L

/D

PIN = %30

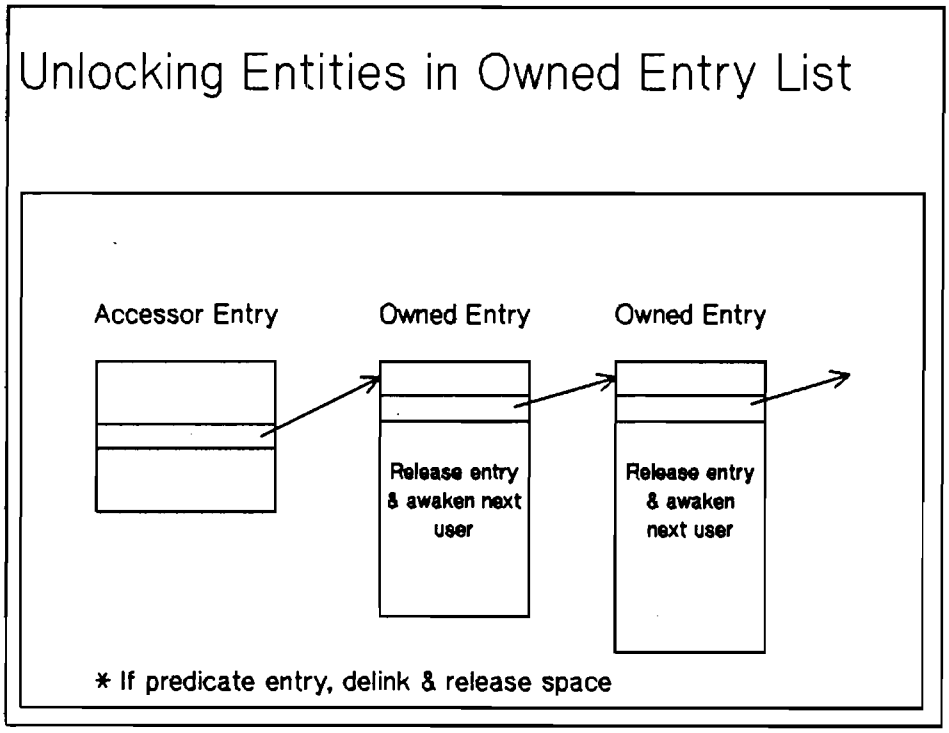
DBH %2156

∴ %10th entry = 8 in PIN hash root table

= Access root table

□ DBULOCK

□ Notes



1. Verify Mode = 1.

2. DBCB preparation.

3. Locate the "Accessor" entry for the user and for every entity on the owner list.
   A. Set "owner user ID" = 0.
   B. If entity is a set or base entry,
      - Set LOCKFLAG = 0.
      - Awake next process in wait queue.
      - Decrement the base entry "lock count".
   C. If entry is a predicate,
      - Locate the associate set entry.
      - Awake any process in wait queue.
      - Delink the predicate from the list.
      - Release space.
      - Decrement set and base entry "lock count".

4. Prepare to exit.

NOTE: DBLOCK and DBUNLOCK do not exchange ULCB information at the beginning or end of the call.

□ Strategy and Guidelines

□ Notes

Access Modes Summary

Access Mode	Type of Access Granted	Concurrent Access Allowed
1	Modify* (With Enforced Locking)	Modify with Enforced Locking
2	Update	Update
3	Modify(Exclusive)	None
4	Modify	Read
5	Read	Modify with Enforced Locking
6	Read	Modify
7	Read(Exclusive)	None
8	Read	Read

* Locking enforced for actual modifications only

☐ Strategy and Guidelines

☐ Notes

Locking and Access Modes

Access Mode 1 - Opened for Shared Modify Access

Action	Intrinsics	Locking Scheme
Modify a Data Entry	DBPUT DBDELETE DBUPDATE	Data Entry, Data Set or Data Base must be successfully locked
Add or Delete from a Master Data Set	DBPUT DBDELETE	Data Set or Data Base must be successfully locked
Update a Master Data Set	DBUPDATE	Data Entry locks are sufficient

BDR0617

HEWLETT PACKARD

Access Mode = 1 - Opened for shared modify access.

Locking is not required for reading data entries with DBFIND and DBGET but this does not protect you from another user simultaneously modifying the same entry which you are reading.

For updates, locks must cover the record before and after the update if the update involves changing the lock item.

□ Strategy and Guidelines

□ Notes

Locking and Access Modes

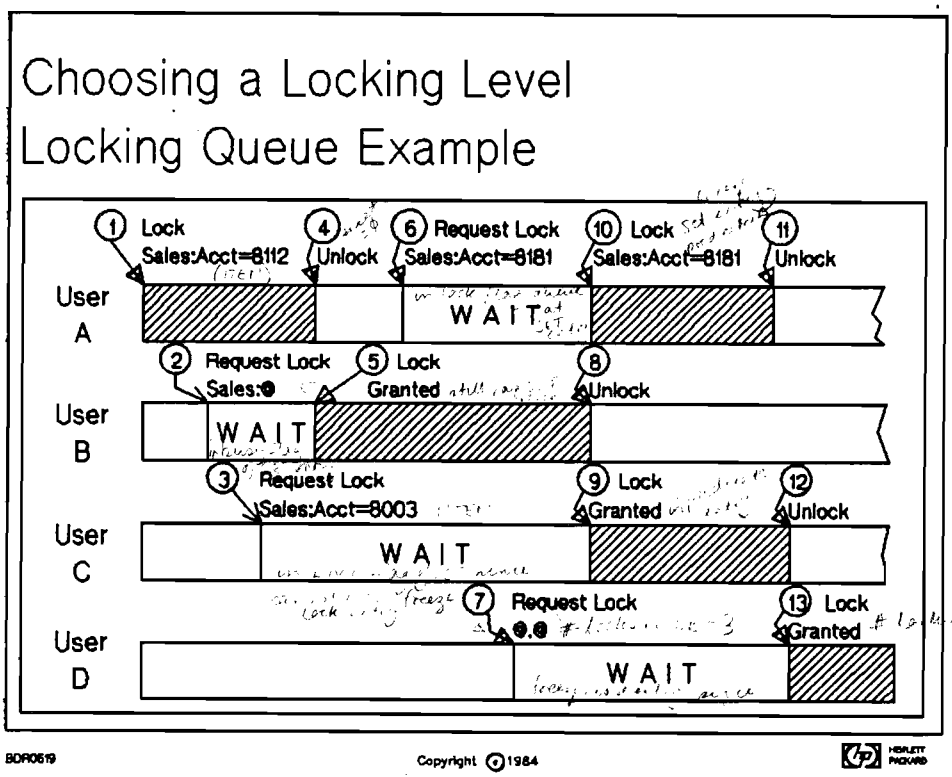
* Access Mode 2 - Opened for Share
  - Locking should be used to coordinate updates with other users
* Locking is not needed with access Modes 3 & 7 since the user has exclusive control of the data base
* Locking is not needed with access Mode 8 since this mode allows concurrent reading only
* For all modes IMAGE does NOT prevent a process from reading data even if another process has it locked

Locking on reads insures no other users modify data while it is being read.

Locking may be used in access modes 2, 4, 5 and 6 to coordinate the reading and modifying sequences.

☐ Strategy and Guidelines

☐ Notes



Program complexity increases the lower the level of locking.

Locking at the set or entry level does not necessarily give the best performance.

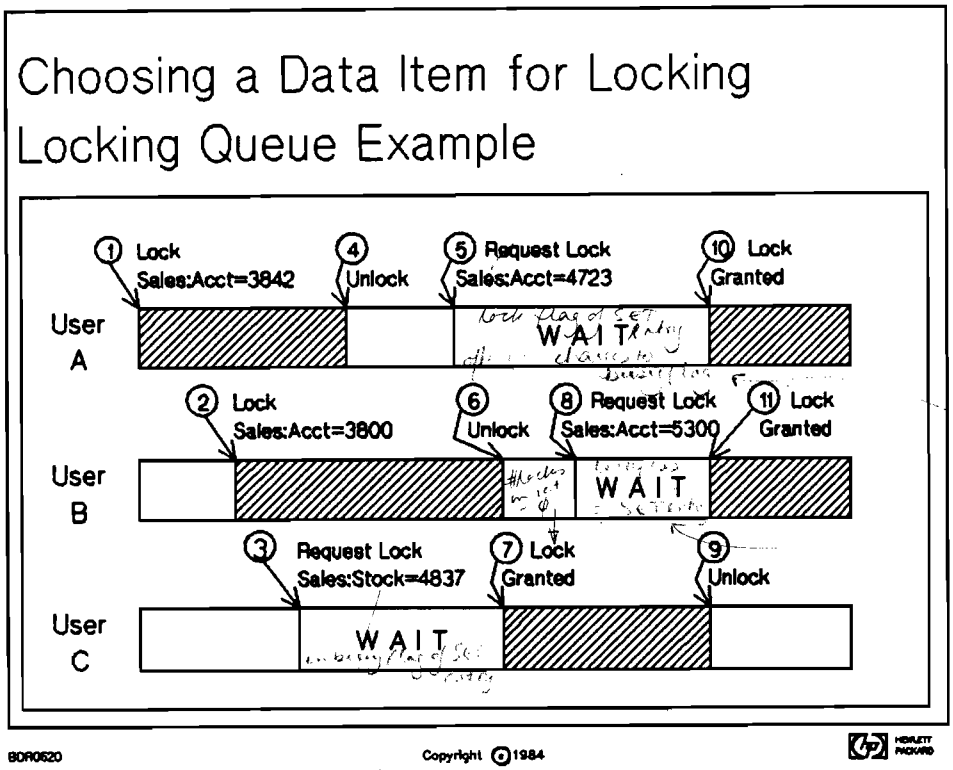
All programs concurrently accessing the data base should generally lock at the same level.

For lengthy transactions (more than 8 IMAGE calls) a user should lock at the data set or entry level.

If user dialog takes place while locking is in effect, data entry level locking should be used.

□ Strategy and Guidelines

□ Notes



All programs sharing the data base concurrently should use the same data item to lock data entries in a particular data set.

IMAGE allows no more than one data item per data set to be used for locking at a given time.

IMAGE does allow multiple values of the data item locked at one time.

☐ Strategy and Guidelines

☐ Notes

Guidelines for Design of Locking Schemes

* CHAINED DBGET CALLS	Lock all data entries in the chain using a single lock descriptor
* SERIAL DBGET CALLS	Lock the data set
* UPDATE A DATA ENTRY (DBUPDATE)	Lock the data entry before calling DBGET to read the data entry, unlock after the update

BDR0521 HEWLETT PACKARD

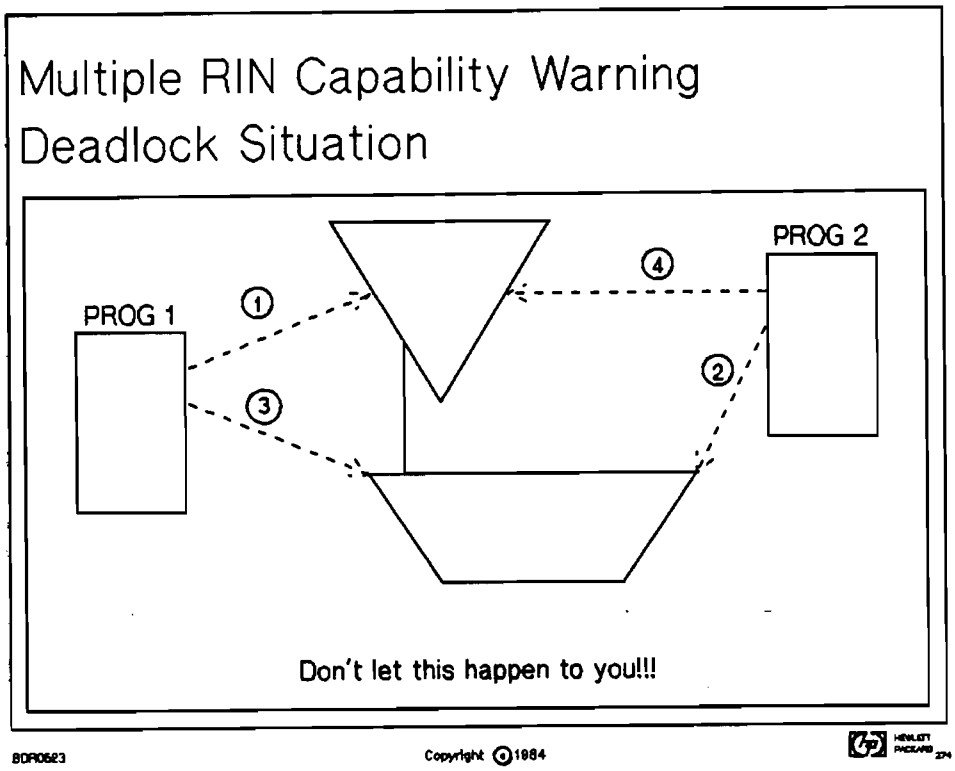
Guidelines for Design of Locking Schemes

* DIRECTED READS (DBGET)	Lock the data set before determining which data entry is needed
* ADD A DATA RECORD TO A DETAIL DATA SET (DBPUT)	Any lock which covers the data entry, preferably use the data entry which is the lock item for that data set
* ADD TO OR DELETE FROM A MASTER DATA SET (DBPUT AND DBDELETE)	Lock the data set or data base

BDR0522 HEWLETT PACKARD

☐ MR Capability

☐ Notes



To insure that two processes do not deadlock, IMAGE does not allow a given process to make a second call to DBLOCK unless locks are cancelled with a call to DBUNLOCK first.

Unless...
The program issuing the multiple calls to DBLOCK was prepared with multiple rin capability.

NOTE: To maintain compatibilty with earlier versions of IMAGE a redundant call may be made to lock the entire data base with DBLOCK Mode 1 or 2 provided the call relates to the same access path.

☐ MR Capability

☐ Notes

Situations Requiring MR Capability

* A program has two or more data bases open simultaneously and wishes to lock part or all of each data base simultaneously

* A program wishes to lock a data base and an MPE or KSAM file simultaneously

* Locks in effect can be seen by the DBUTIL SHOW LOCK command

Module 7.1

Transaction Logging and Intrinsic Level Recovery

QUIZ

1. When updating an item (not a search item) that is defined as the lock item, why is it necessary to lock both the old and new values? lock old value [illegible] change [illegible] and new value [illegible]

2. Why should you use same lock item whenever possible? [illegible] to speed up hardware locks

3. What is the danger of using MR capability? Deadlock [Coolstart/warmstart]
(1) lock in correct order (2) [illegible]

4. What are the three lock entry types used by IMAGE? DB, data set, data item

5. What is the purpose of the lock flag? to prevent a lock at a particular level or prevent locks below that level. [illegible]

6. What is the purpose of the busy flag? [illegible] for locks to be released

7. If a record level DBLOCK succeeds, does that mean the record exists? Explain. No, [illegible]

8. Why should you lock the data set or data base before adding to or deleting from a master data set? maintaining [illegible]

Module 7.2

Module Outline

A. Transaction Logging

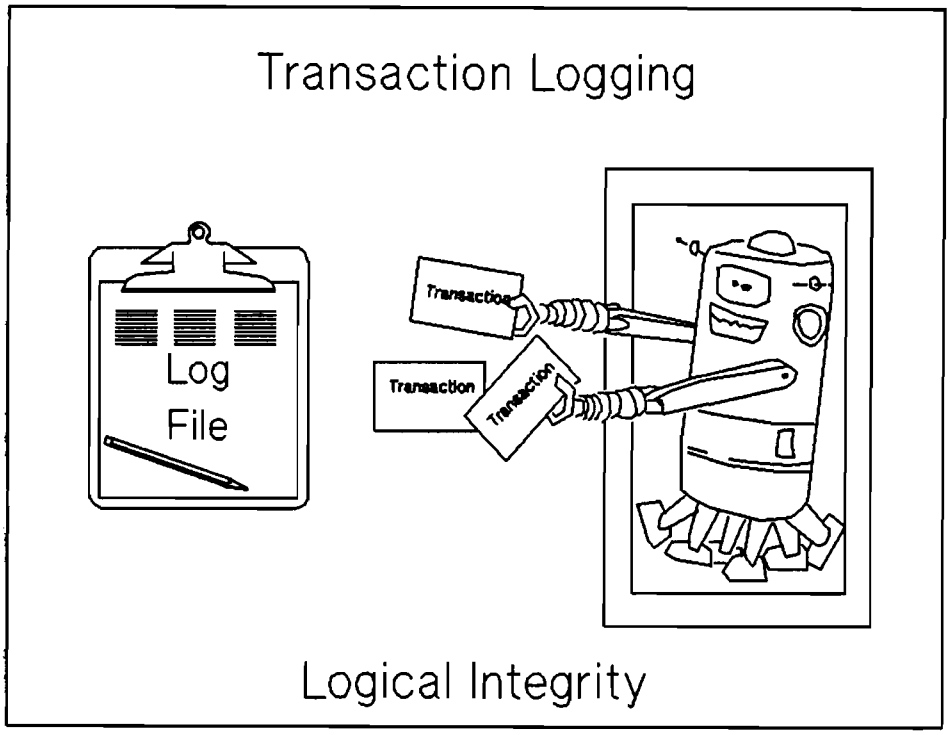
0. Introduction
1. Benefits
2. How it Works
3. MPE Log Buffer
4. MPE Record Formats
5. Sharing Log Files

B. Intrinsic Level Recovery

0. Introduction
1. ILR Log File
2. ILCB
3. ILR Control Block
4. How it Works
5. Maximum Buffers Modified
6. Impact of ILR on Performance

☐ Transaction Logging

☐ Notes



Transaction logging and recovery allows transactions to be logged to a log file on either magnetic tape or disc. Unlike Intrinsic Level Recovery which ensures the structural integrity of an IMAGE data base, transaction logging ensures the logical integrity of an IMAGE data base.

□ Transaction Logging

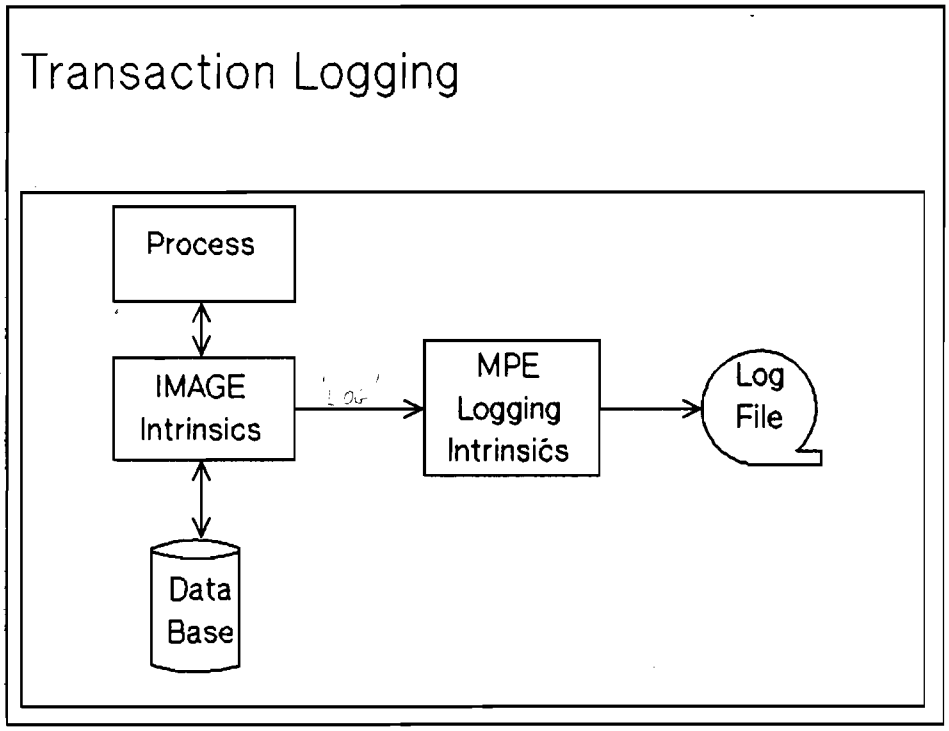
□ Notes

Benefits of Logging VS Costs of Logging

Benefits	Costs
o Provides audit trail	o Reduced throughput (application dependent)
o Ability to recover transactions	o May require additional memory
o Performance information	o Dedicated tape drive or disc space consumption
	o Startup after crash more complex

□ Transaction Logging

□ Notes



- DBPUT - copy of record being added
- DBUPDATE - copy of old and new record
- DBDELETE - copy of deleted record
- DBOPEN - record of timestamp of last DBSTORE, user identifier, log identifier, name, group and account of user, data base, program

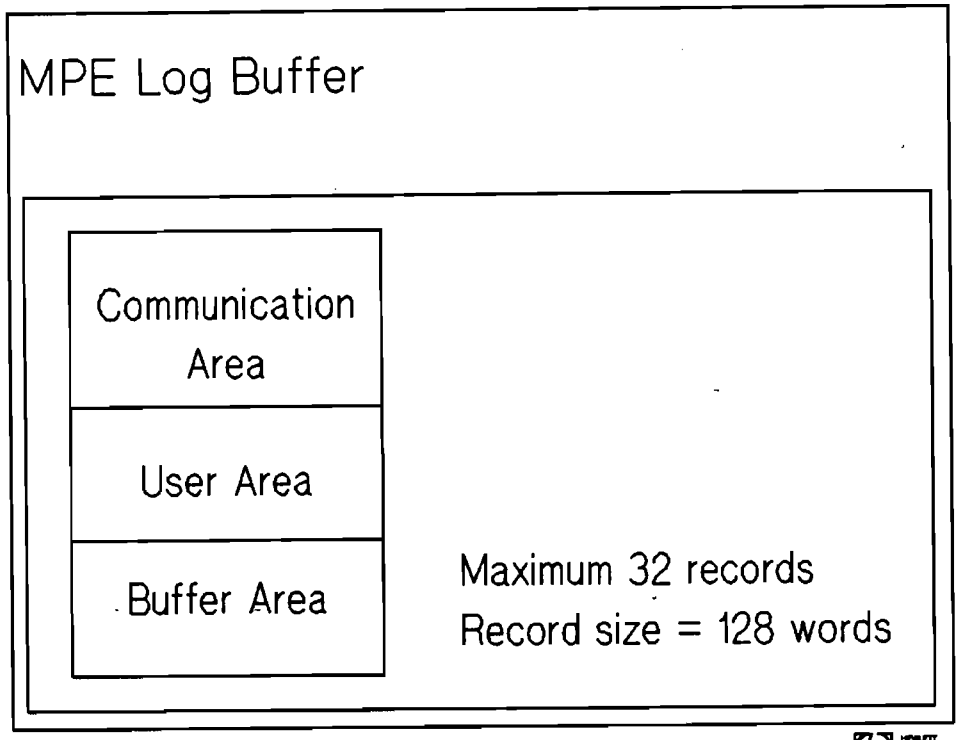
If the transaction is large (over 238 bytes of data), WRITELOG breaks the transaction into a main record and continuation records. User logging holds the transactions temporarily in an extra data segment and periodically flushes the transactions to a disc file. If logging to a tape is desired, the disc file is periodically copied to tape.

The following error codes are returned by IMAGE in the status area when logging errors are encountered:

- -110 OPENLOG failure (DBOPEN only)
- -111 WRITELOG failure
- -112 CLOSELOG failure (DBCLOSE only)

□ Transaction Logging

□ Notes



When transaction logging is enabled, the following intrinsics are logged to the log file:

- DBPUT
- DBUPDATE
- DBDELETE
- DBOPEN (Modes 1-4)
- DBCLOSE
- DBBEGIN
- DBEND
- DBMEMO

□ Transaction Logging

□ Notes

MPE Log Record Formats

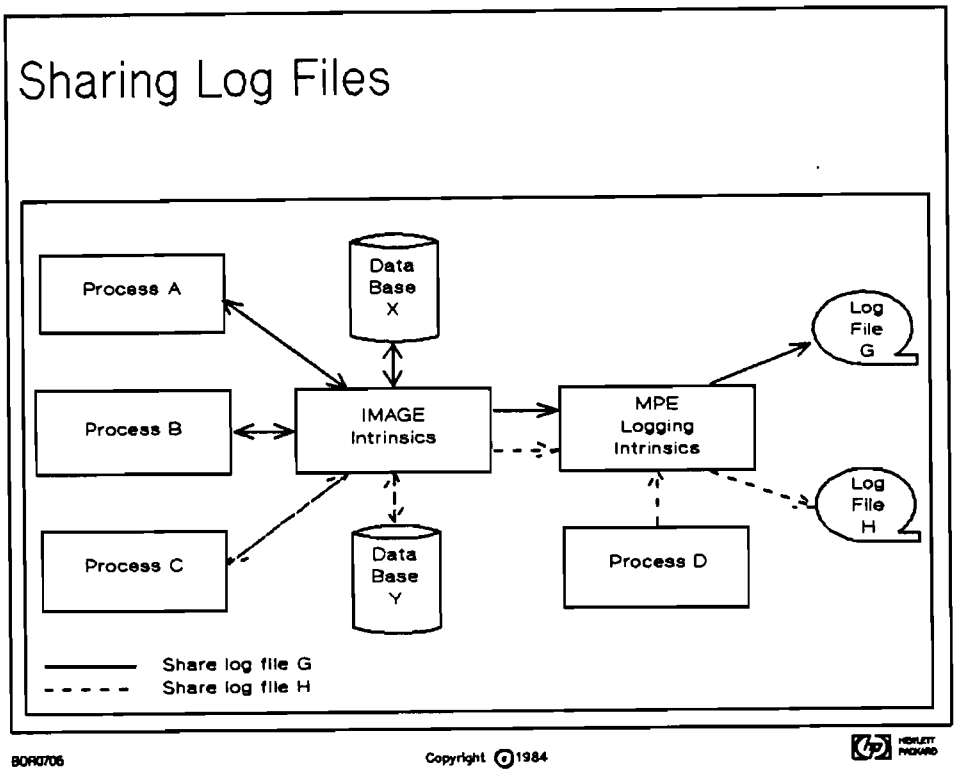
Word	
0	Record Number (2 words)
2	Checksum
3	Subsystem Identifier (1st byte)
3	Log Record Code (2nd Byte)
4	Time
6	Date
7	Logging Identifier (nonexistant for CRASH Log Number for WRITELOG)
8	User Buffer Length (WRITELOG only)
9	User Buffer Area (WRITELOG only)
11	Log Number (OPENLOG, CLOSELOG only)
12	User Name, Group, Account (OPENLOG, WRITELOG only)
24	PIN# (OPENLOG, WRITELOG only)

BDR0705

HEWLETT PACKARD

□ Transaction Logging

□ Notes

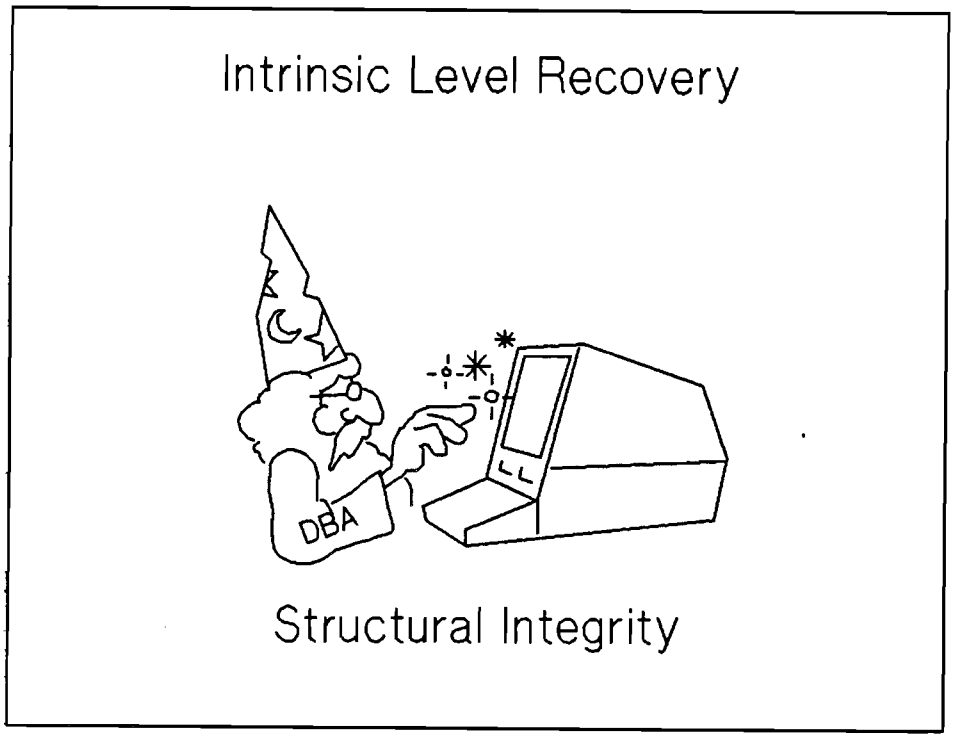


Multiple data bases may share the same log file. The limiting factor on sharing a log file is that only 128 user processes may share a log file at one time. Read access users are not counted in the 128, however users who open two data bases with update access are counted twice.

A WARMSTART following a system failure will attempt to clean-up any log files that were open at the time of the crash. The clean-up procedure appends any records left in the log system disc buffer file to the end of the log file. For tape logging, it copies any records left in the disc buffer file to the end of the logtape. In either case, any records left in the memory buffers (extra data segment) will be lost.

☐ Intrinsic Level Recovery

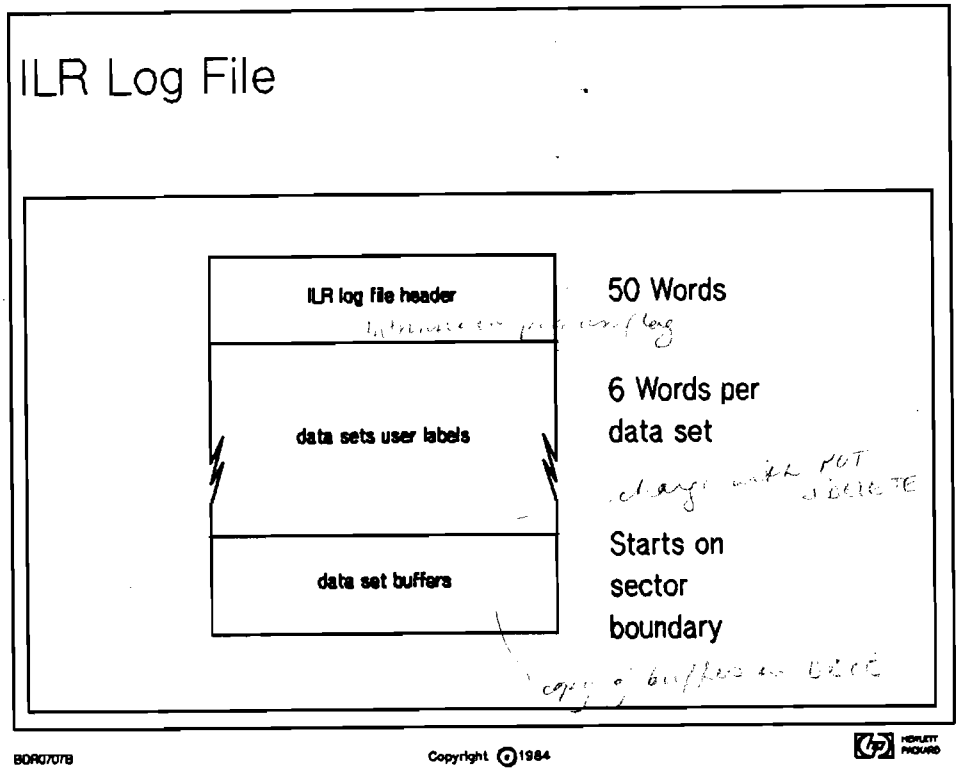
☐ Notes



Intrinsic Level Recovery (ILR) allows IMAGE to automatically recover and restore any broken chains, thus ensuring the structural integrity of an IMAGE data base. ILR was released with IMAGE version B.04.00, and is compatible with Q-MIT C.01.00, D.01.00 or later versions of MPE.

□ Intrinsic Level Recovery

□ Notes



When ILR is enabled, IMAGE maintains an ILR log file where intrinsics which modify the media record pointers (DBPUT and DBDELETE) are automatically logged. DBUPDATE is not logged since it only affects non-key items.

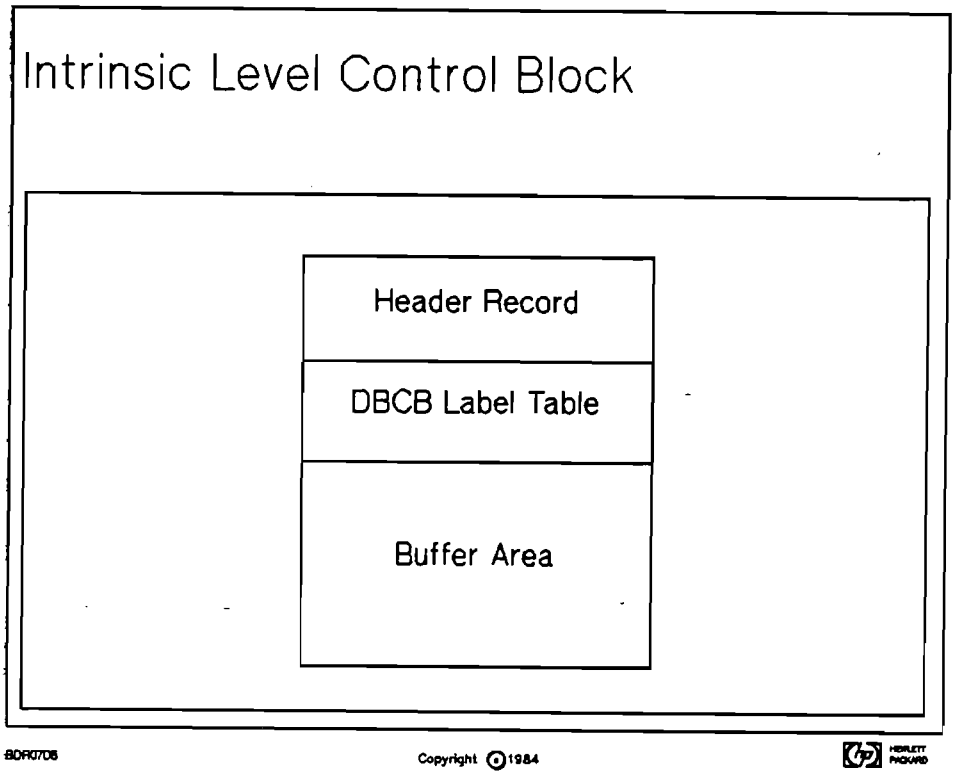
At the time ILR is ENABLED (using DBUTIL), DBUTIL builds the ILR log file, and sets a flag in the data base root file to indicate that the ILR log file is enabled. DBUTIL also sets a creation date and time stamp in both the ILR log file and the data base root file. The ILR log file is a privileged file who`s name is derived by appending "00" to the root file name.

The intrinsic-in-progress flag is set when a DBPUT or DBDELETE begins execution. Both the header and original data set user labels are posted to the ILR log file before any data sets are modified by the intrinsic. The data set buffer will hold as many buffers as may be modified by one intrinsic.

ILR Log File Header:
- intrinsic-in-progress flag
- intrinsic type
- data set being modified

☐ Intrinsic Level Recovery

☐ Notes

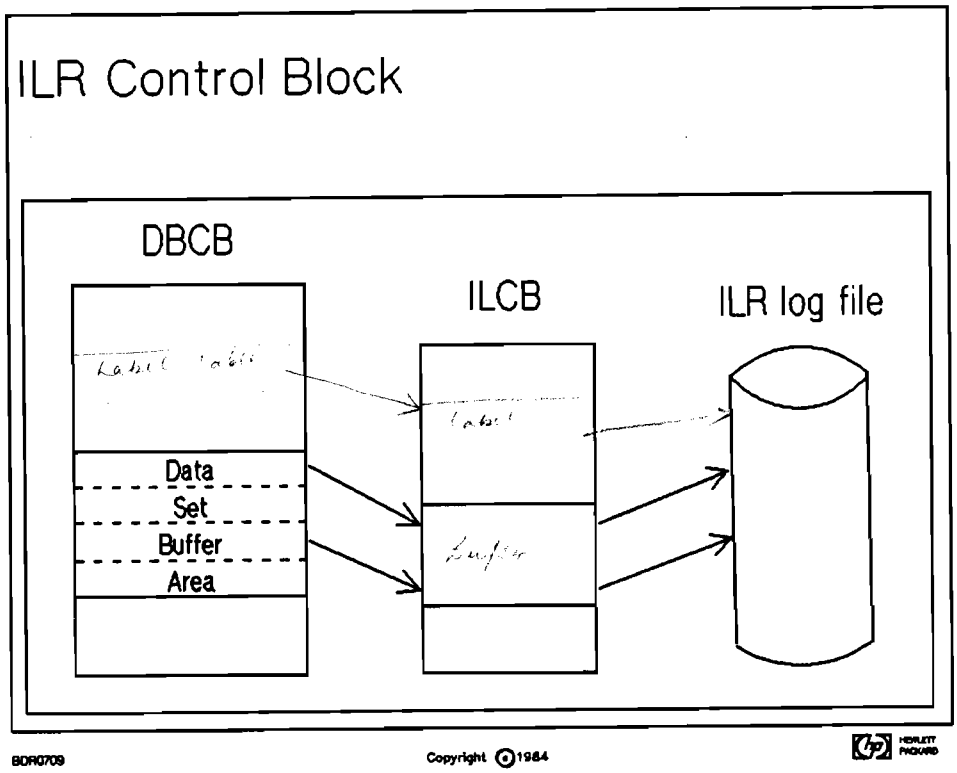


For a data base enabled for ILR, a single extra data segment, the Intrinsic Level Control Block (ILCB), is allocated with the first DBOPEN of the data base. The ILCB is used as an intermediate staging area for the buffers that will be modified by each DBPUT or DBDELETE.

The ILCB size depends on the structure of the DBCB. It should hold as many buffers as can be modified in the DBCB before posting to the data sets. The DBCB should hold at least as many buffers as can be modified (by a DBPUT or DBDELETE) to minimize buffer rollover.

□ Intrinsic Level Recovery

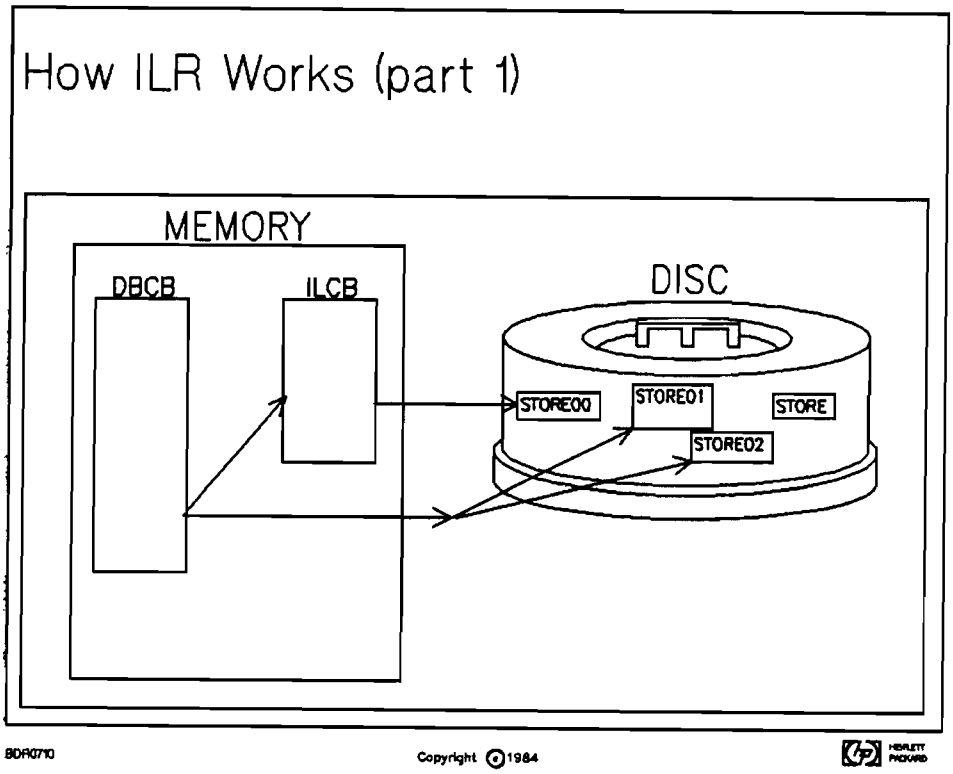
□ Notes



When a DBPUT or DBDELETE executes, the current data entries to be modified are read into the Data Base Control Block (DBCB) as buffers. These buffers are immediately posted to the ILCB (before they are modified in the DBCB). They reflect the state of the data base before the intrinsic began execution. The buffers are then modified in the DBCB. When the modified buffers in the DBCB are ready to be posted to the data base, the unmodified buffers are posted to the ILR log file. After the modified buffers have been completely posted to the data base, the intrinsic-in-progress flag in the ILR log file is reset to not-in-progress. The ILR log file is overlayed so that the only information found there is for the intrinsic in progress, not for all of the intrinsics that have occurred in the data base.

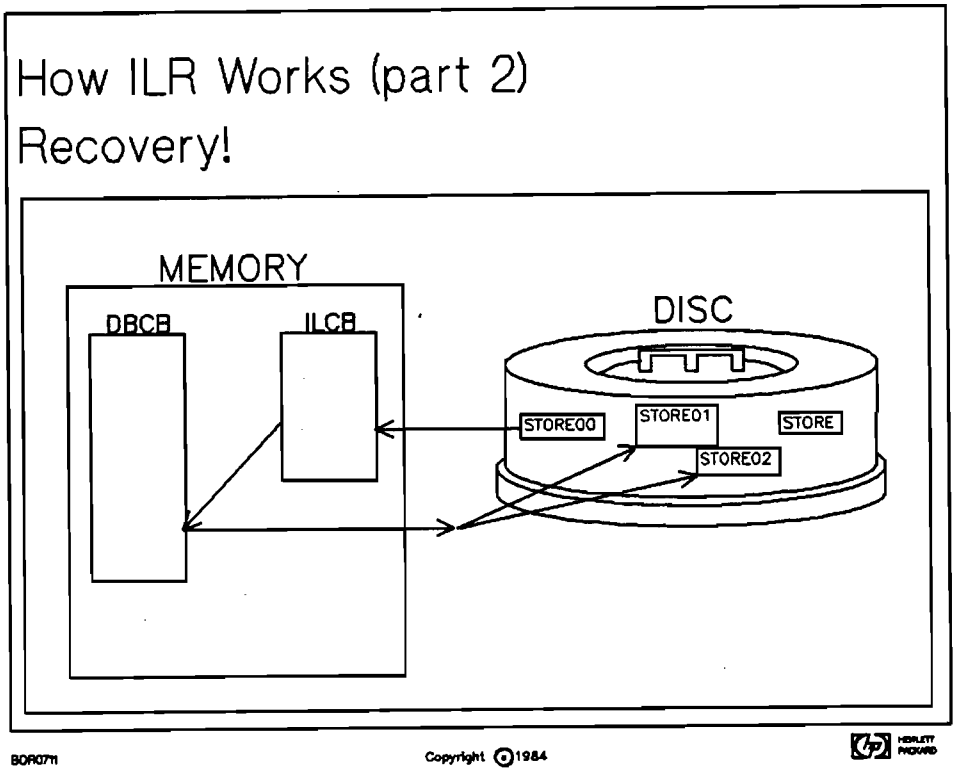
☐ Intrinsic Level Recovery

☐ Notes



□ Intrinsic Level Recovery

□ Notes



When a data base that is enabled for ILR is first opened using DBOPEN, the intrinsic-in-progress flag is examined to determine if the execution of a prior intrinsic was interrupted during the last access of the data base. If the flag is on, the before modification buffers in the ILR log file are posted to the data sets. Thus, the internal structure of the data base is returned to the state it was in before the interrupted transaction began executed. Structural integrity is therefore restored to the data base.

☐ Intrinsic Level Recovery

☐ Notes

Maximum Buffers Modified

Master Data Sets:

4 (automatic or manual)

Detail Data Sets:

(4 * Autos) + (3 * Manuals) + 1

The maximum number of buffers that can be modified must be determined to determine the overhead incurred with ILR. This number depends on the structure of the data base. For master data sets, either automatic or manual, the maximum number of buffers that can be modified is 4. For detail data sets the maximum is equal to 4 times the number of paths from an automatic master pointing to it, plus 3 times the number of paths from a manual master pointing to it, plus 1:

Max`mod`buffs = (4 * Autos) + (3 * Manuals) + 1

□ Intrinsic Level Recovery

□ Notes

Impact of ILR on Performance

Shared privileged mode file (ILR log file)

Shared extra data segment (ILCB)

Disc I/O to post the ILCB to the ILR log file

3 minimum

BDR0713

The number of disc I/Os incurred for ILR is as follows. At the beginning of the intrinsic, one disc I/O records date, time, data set number, intrinsic-in-progress flag, and data set user labels. The buffers are then posted, tacking one or more I/Os, depending on the relationship of the maximum number of DBCB buffers that were allocated. For example, if 15 buffers have to be modified, but the DBCB can hold only 8 buffers, some of the buffers will have to be posted before they are overlayed. Performance can be improved by allocating sufficient DBCB buffers using the DBUTIL SET command. When all of the buffers are posted to the ILR log file, and the data base has been updated, one additional I/O has to be performed to turn off the intrinsic-in-progress flag in the ILR log file.

The minimum number of additional disc I/Os required for DBPUTs and DBDELETEs, therefore, is 3. The maximum will depend on the maximum number of buffers that need to be modified, and the number of times the DBCB buffer area has to be reused. The maximum number of buffers is only used when all entries that need to be modified reside in different physical disc blocks.

Module 8.1

IMAGE with MPE V/P and MPE V/E

Module 8.2

Module Outline

A. MPE V/P

1. Overview of Disc Caching
2. Commands
3. IMAGE and Disc Caching

B. MPE V/E

1. Overview of MPE V/E Changes
2. DST, PCB Expansion
3. Effects on IMAGE

□ MPE V/P

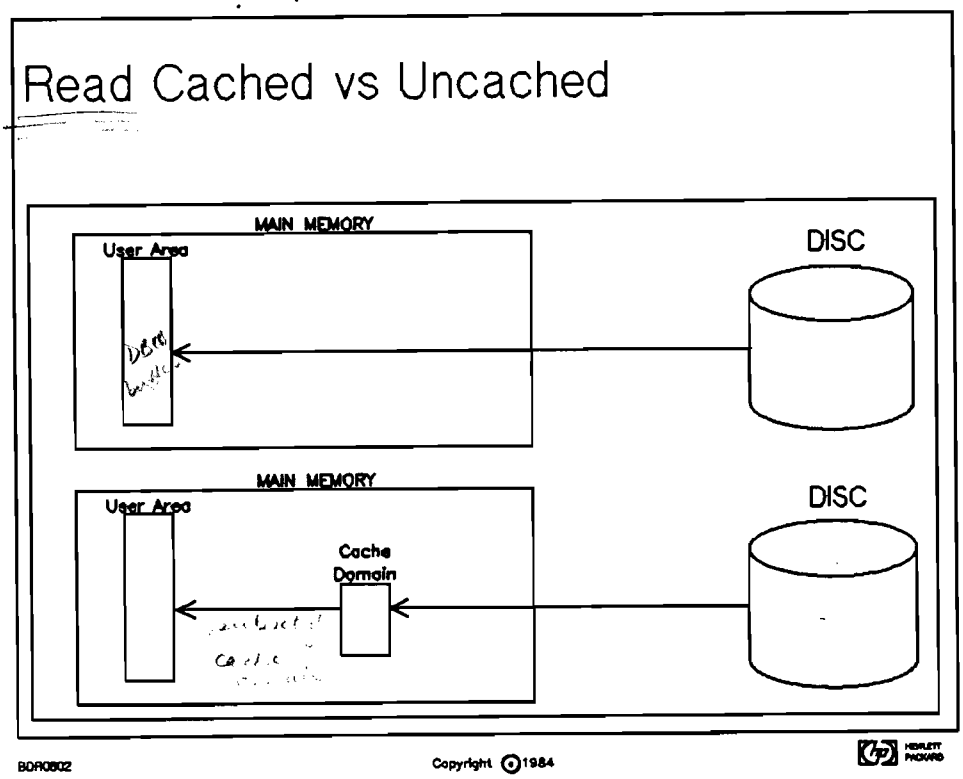
□ Notes

Disc Caching

* Excess memory used so that disc records may remain in memory
* Definitions
    1) Cache Domain - Extra data segment reserved for caching
    2) Physical I/O - I/O from/to disc
    3) Logical I/O - CPU move from/to cache domain
    4) Hit - Read/write to a cache domain already in memory
    5) Miss - Read/write to a cache domain which must be first brought into memory
* Objective is to cut the number of physical I/O's to disc

□ MPE V/P

□ Notes



User issues a read request . . .

Uncached

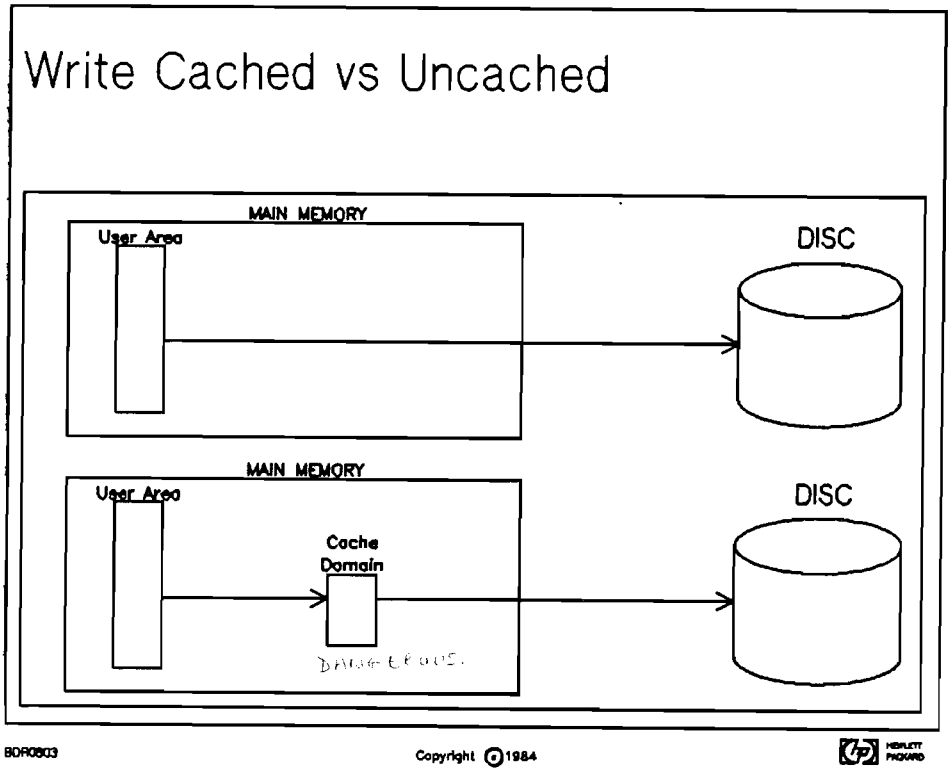
1. User area checked to see if record exists in a buffer.
2. If not in user area, go to disc and bring in a block of records to user area buffer.

Cached

1. User area checked to see if record exists in a buffer.
2. If not in user area, check cache domains for record. If in a cache domain, bring into user area buffer (saving a disc I/O!).
3. If not in a cache domain, go to disc and bring in a new domain.

□ MPE V/P

□ Notes



User issues a write request . . .

Uncached

1. Write record from buffer to disc - process impedes until write is complete (unless operating in output deferred mode).

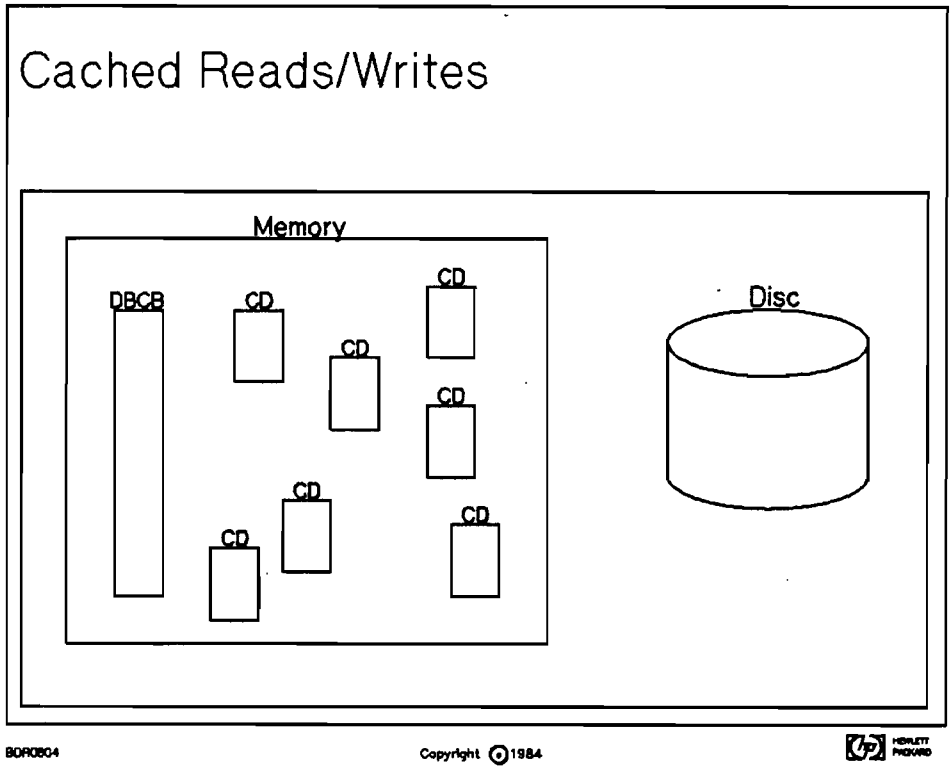
Cached

1. Look for cache domain containing record. If one is found, check if a write from the domain has been issued.
   a. If the record is in a cache domain with prior write request, domain is first flushed to disc, impeding the user issuing the current request. When the write has been completed, modify domain.
   b. If the record is in a cache domain without a prior write request, domain is modified.
   c. If the record is not in a cache domain (write miss), create domain the size of the data being transferred.
2. Issue write request. If BLOCKONWRITE=yes, then impede, otherwise return to user.

NOTE: Partial writes will cause a process stop - new cache domain must be created and will overlay old one.

□ MPE V/P

□ Notes



Cache domains are maintained by MPE through the Cache Directory Table (CDT). This table is itself an extra data segment.

□ MPE V/P

□ Notes

When to Use Disc Caching

1) CPU utilization is low (OPT/3000 w/MPE IV):
   * 4X systems --> CPU <= 60% busy
   * 6X systems --> CPU <= 70% busy

2) System has excess Main Memory (OPT/3000 w/MPE IV):
   * Memory Management I/O <= 5% of total I/O

3) System has minimum Main Memory size:
   * Series 39/42 = 2Mb
   * Series 48 = 3Mb
   * Series 68 = 4Mb

□ Notes

When Not to Use Disc Caching

1) CPU utilization is high (OPT/3000 w/MPE IV):
   * 4X systems --> CPU >= 70% busy
   * 6X systems --> CPU >= 85% busy

2) System has no excess Main Memory (OPT/3000 w/MPE IV):
   * Memory Management I/O >= 10% of total I/O

3) System does not have a sufficient amount of Main Memory:
   * Series 39/42 < 2Mb
   * Series 48 < 3Mb
   * Series 68 < 4Mb

□ MPE V/P

□ Notes

Caching Objectives

* Maximize Read Hits
* Minimize Process Stops
  - Cache domain or user stack not in memory (memory pressure)
  - Logical write contention (write hits)
* Minimize Write Hits

These were the objectives of the caching design. If your system is succeeding in all of these then the caching is well suited to your system. Otherwise, you should investigate why caching is not performing well and reconsider its use.

□ MPE V/P

□ Notes

:SHOWCACHE

* Gives information on disc caching
    * Read Hit %
    * Write Hit %
    * Read %
    * Number of Process Stops
    * Number of Cache Requests
* Only for those discs using caching

CACHE REQUESTS = read + write requests to caching
READ HIT% = READ HITS / REQUESTS (READ)
WRITE HIT% = WRITE HITS / REQUESTS (WRITE)
READ% = READS / CACHE REQUESTS

PROCESS STOPS are the number of times processes stopped in order to for a cache I/O request to complete. The process stops are derived from one of two main reasons:

1. The cache domain or user stack is not in memory (memory pressure).
2. A write hit results in contention for logical I/O to a cache domain.

Data overhead = size cache XDS + # cache domains * (256 bytes). The 256 bytes is the overhead from the region header; they require 1 sector because thay always fall on a sector boundary.

% user I/Os eliminated = read hit % * read %.

□ MPE V/P

□ Notes

Controlling Disc Caching

:CACHECONTROL – Sets caching parameters for entire system

* Sequential fetch quantum
* Random fetch quantum
* BLOCKONWRITE mode

FSETMODE(filenum,modeflags) – sets modes on a file-by-file basis

* BLOCKONWRITE
* Serial Write Queue

:CACHECONTROL is a new command which allows the user to set the fetch quantums and BLOCKONWRITE. These apply globally to the system.

FSETMODE intrinsic can be used to control individual files.

The fetch quantums are used in caching strategy; to determine size of the cache domains. The BLOCKONWRITE flag and the Serial Write Queue are used to guarantee write commitment to disc and write seriality. First we will examine the fetch quantums.

□ MPE V/P

□ Notes

Fetch Quantum Strategy

* Determining cache size for read misses
* Strategy dependent on access type and on random and sequential quantums

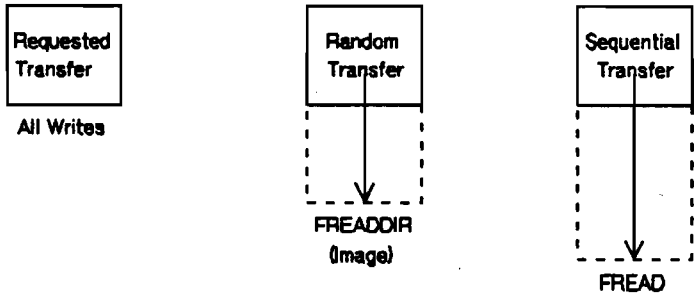


IMAGE always uses FREADDIR on a serial read, so always using Random fetch quantum

For write misses, the cache domain is always the size of the data being transferred.

For read misses, a round up strategy is applied; a larger cache than that requested is created to maximize the read hits. The size of the round up is dependant on the type of file being accessed (random, sequential), which is passed to caching by the fie system. The size of the domain is the highest even multiple of the requested size less than the appropriate fetch quantum.

Of course, if the requested size is larger than the fetch quantum, the cache size is the size of the requested transfer. In other words, the roundup strategy never decrease the size of the cache domain.

FREAD_IR request 15 sectors use Random fetch quantum which may be 96 sectors

Round up is highest multiple requested sectors < than 96 fetch quantum.

ie 90 sectors (15×6) is actual transfer.

□ MPE V/P

□ Notes

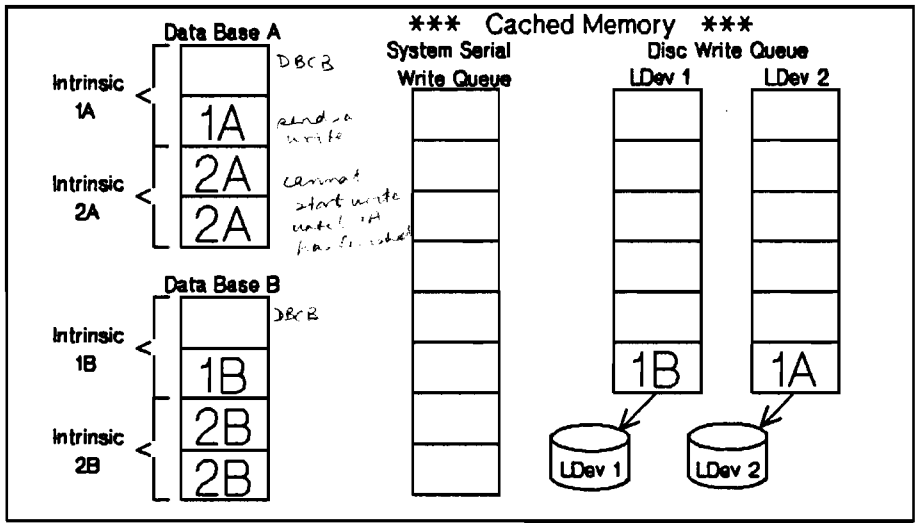
Image and Disc Caching

* Image will use disc caching if started by operator for the disc(s) that contain a data base or part of a data base
* System Wide Serial Write Queue used with ILR and Logging to insure chronological order of writes
* BLOCKONWRITE may be used to further insure integrity

□ MPE V/P

□ Notes

IMAGE and Disc Caching



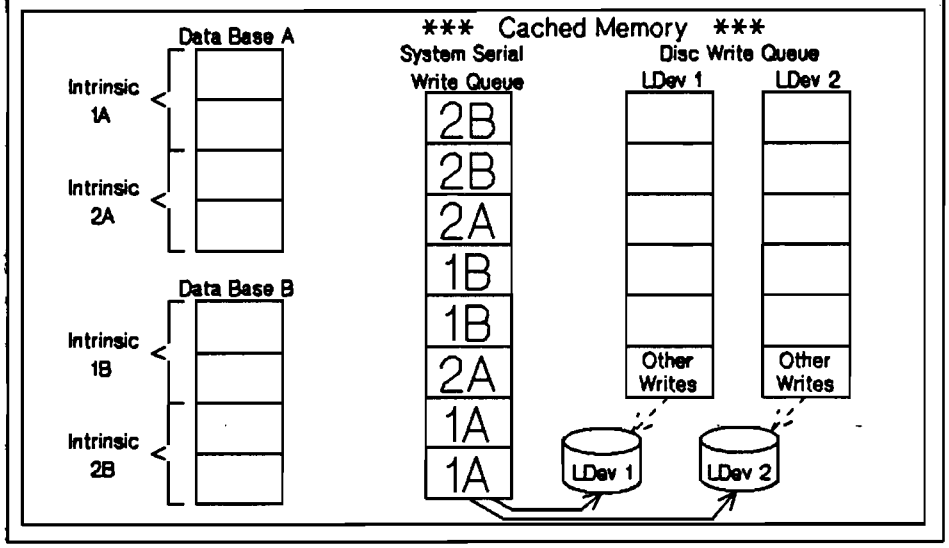
Model 1

- BLOCKONWRITE = YES
- Posting to disc occurs while DBCB wait field is held. Only 1 I/O per data base in progress.

□ MPE V/P

□ Notes

IMAGE and Disc Caching



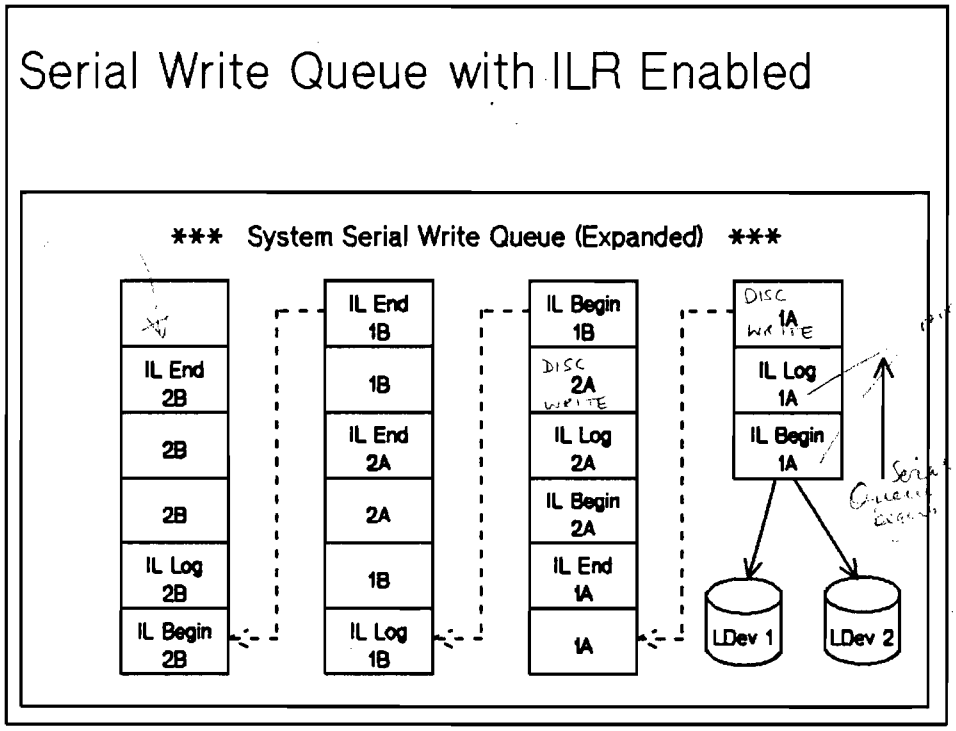
Model 2

- BLOCKONWRITE = NO ILR or logging enabled.
- ILR will maintain the structural integrity of the data base using the system serial write queue to sequence disc writes in chronological order.
- Writes to the ILR file wait for previous writes to ILR file wait for previous writes to ILR file to complete.
- Writes serialized for all databases with any logging facility enabled.

□ MPE V/P

□ Notes

Serial Write Queue with ILR Enabled

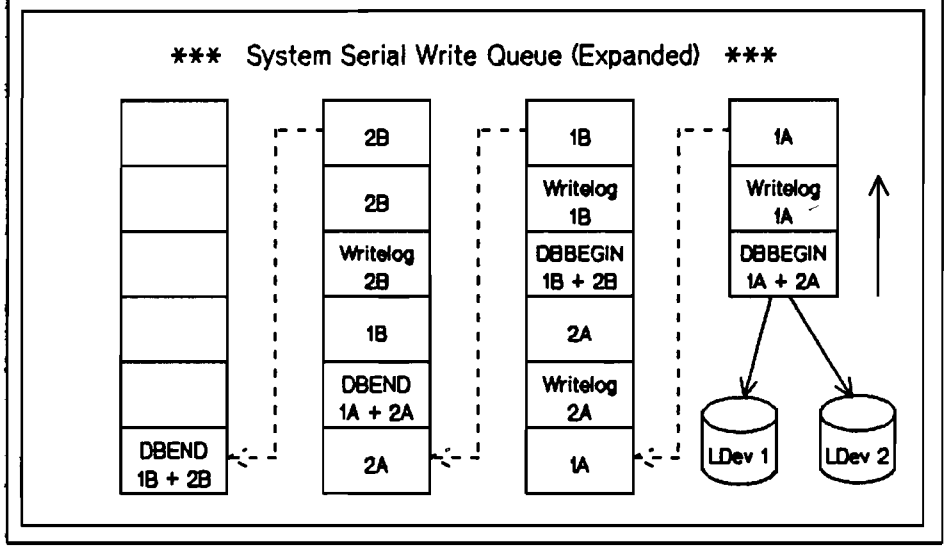


- Serialization ensures ILR writes will complete in the order required to maintain physical integrity of the data base.
- Caching synchronization mechanisms do not allow more than 1 pending write to the same area on disc. Since ILR writes the same disc area, should never be more than 1 behind.

□ MPE V/P

□ Notes

Serial Write Queue with Logging Enabled

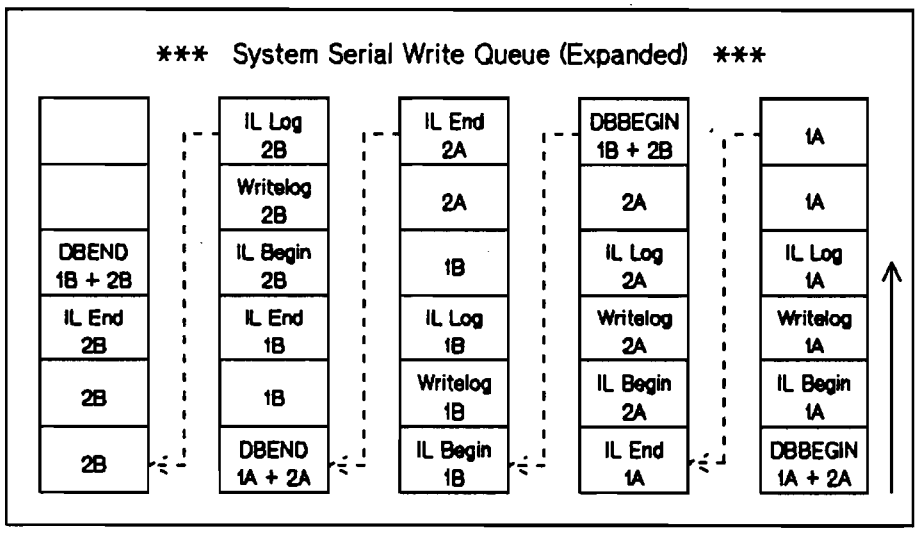


- DBEND should be used when all writes for the transaction must complete before the program continues execution.
- DBEND causes 1 blocked write per transaction. This blocked write will wait for the other writes in the serial queue to complete before it completes.

□ MPE V/P

□ Notes

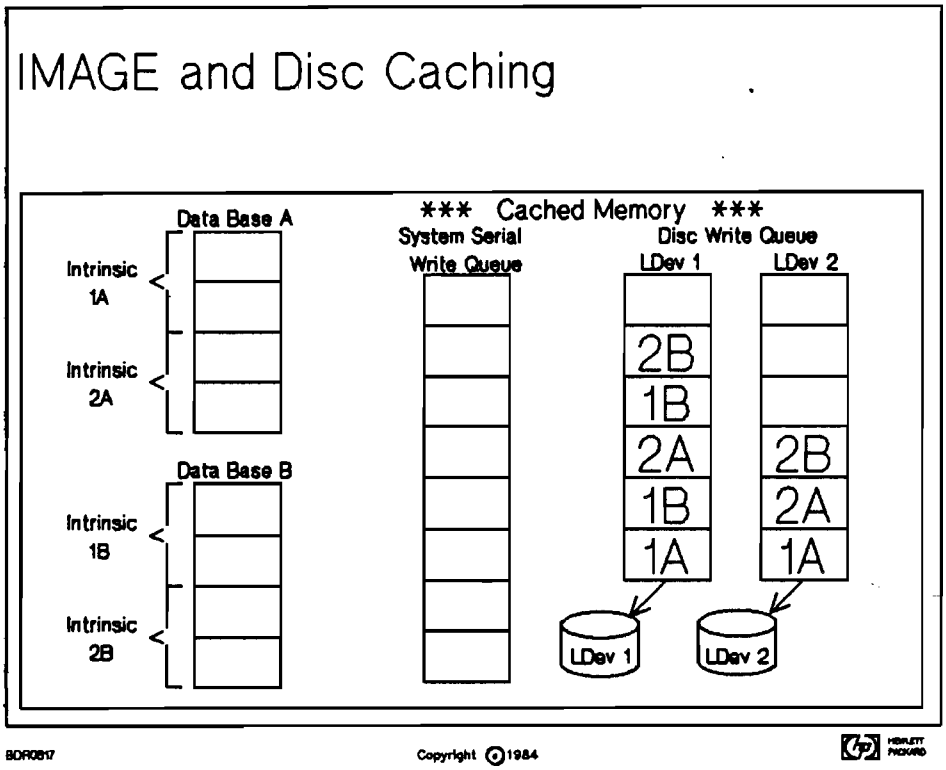
Serial Write Queue with ILR and Logging Enabled



- DBEND while logging causes 1 blocked write per transaction.
- ILR causes 3 unblocked writes per DBPUT or DBDELETE.

□ MPE V/P

□ Notes



Model 3

- BLOCKONWRITE = NO
  ILR and logging not enabled.
- More intrinsics are allowed to execute concurrently; a system interruption will now cause multiple structural inconsistencies.

Module 8.20

□ Notes

IMAGE Changes - Introduction

* No External Changes to IMAGE
* Impact on internal structures
* Affects DBOPEN, DBCLOSE, and DBLOCK

MPE V/E Changes that Affect IMAGE

* DST Expansion
  - From 1024 Entries to 4096 Entries
  - 12 bits now needed to represent DST # (formally 10 bits)
* PCB Expansion
  - From 256 Entries to 1024 Entries
  - 10 bits needed to represent PIN (formerly 8 bits)

□ MPE V/E

□ Notes

IMAGE Changes

For versions B.04.00 and later...

1. Wait Fields
2. User ID
3. System Data Base Control Block (SDBCB)

□ MPE V/E

□ Notes

IMAGE Changes

Wait Fields

2 Word Wait Field

PIN of Current Owner	Queue Count
Queue Tail	Queue Head

MPE IV & Old IMAGE
(B.03.10 and earlier)

4 Word Wait Fields

PIN of Current Owner	Queue Count
Queue Tail	Queue Head
Unused	
Unused	

MPE IV & New IMAGE
(B.04.00 and later)

PIN of Current Owner
Queue Head
Queue Tail
Queue Count

MPE V/E & New IMAGE
(B.04.00 and later)

BDR0821

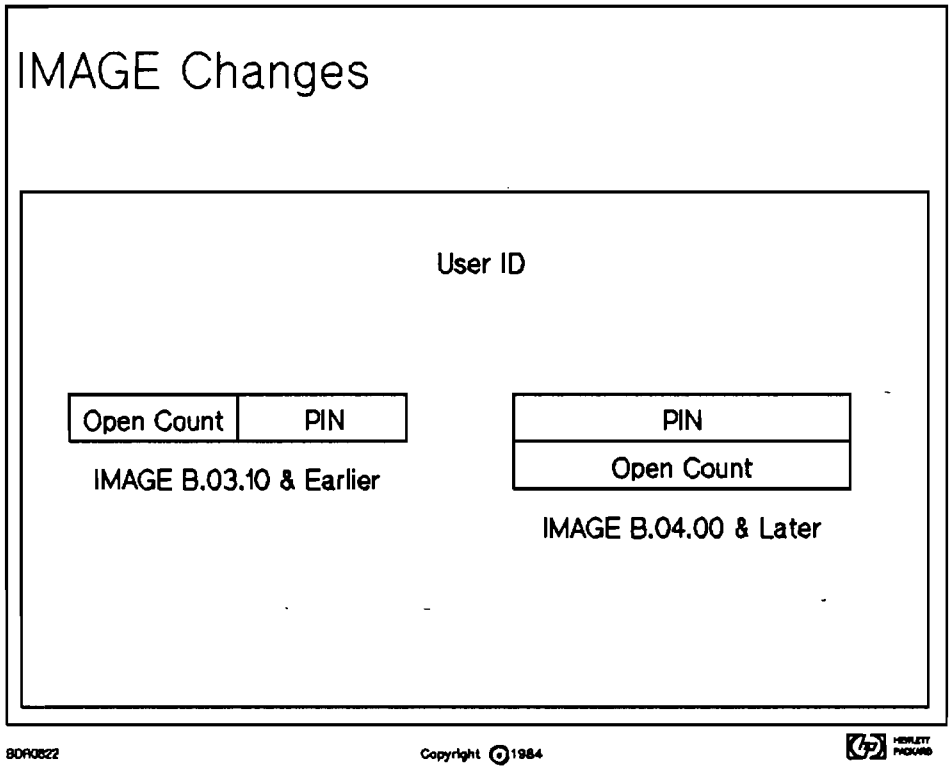
HEWLETT PACKARD

Effects on:

- DBCBWait Field
- Base entry
- Set entries
- Predicate entries

□ MPE V/E

□ Notes



PINs have expanded from one byte to one word (to allow for more processes).

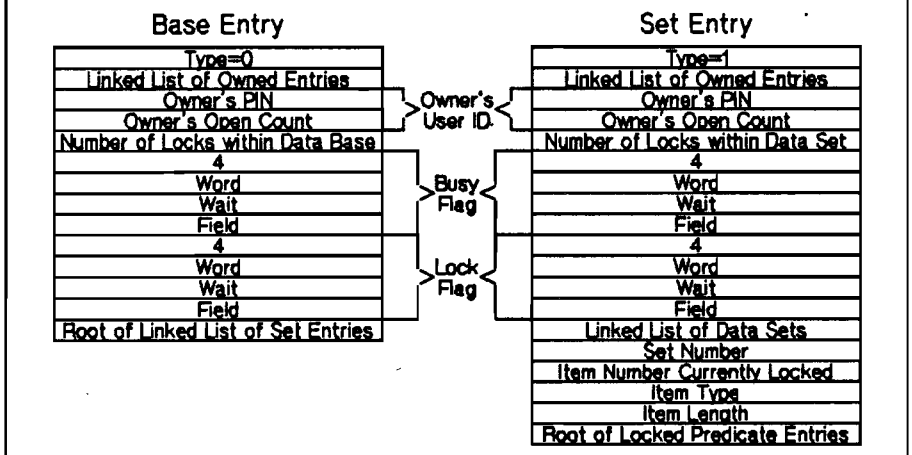
User ID now 2 words long.

Effects on:
- Accessor Entry
- Base, set, predicate entries

□ MPE V/E

□ Notes

New Table Layouts



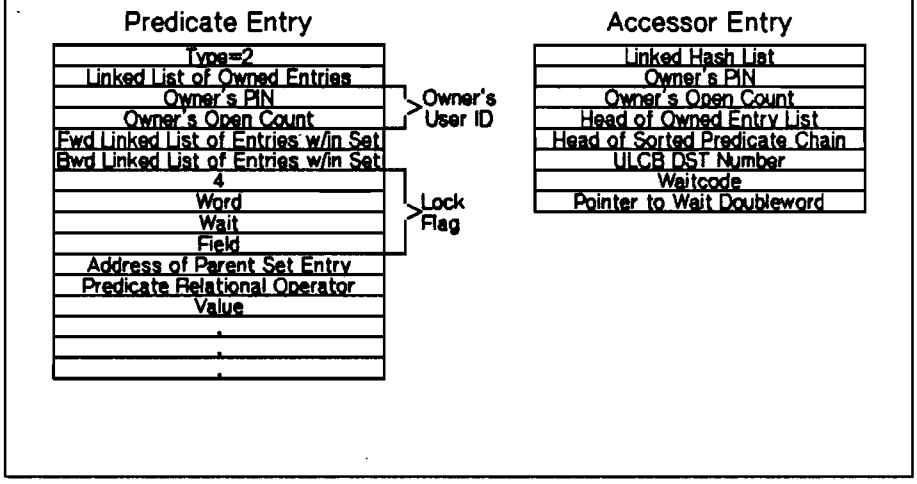
BDR0823

IMAGE versions B.04.00 and later.

□ MPE V/E

□ Notes

New Table Layouts



BDR0824

IMAGE versions B.04.00 and later.

□ MPE V/E

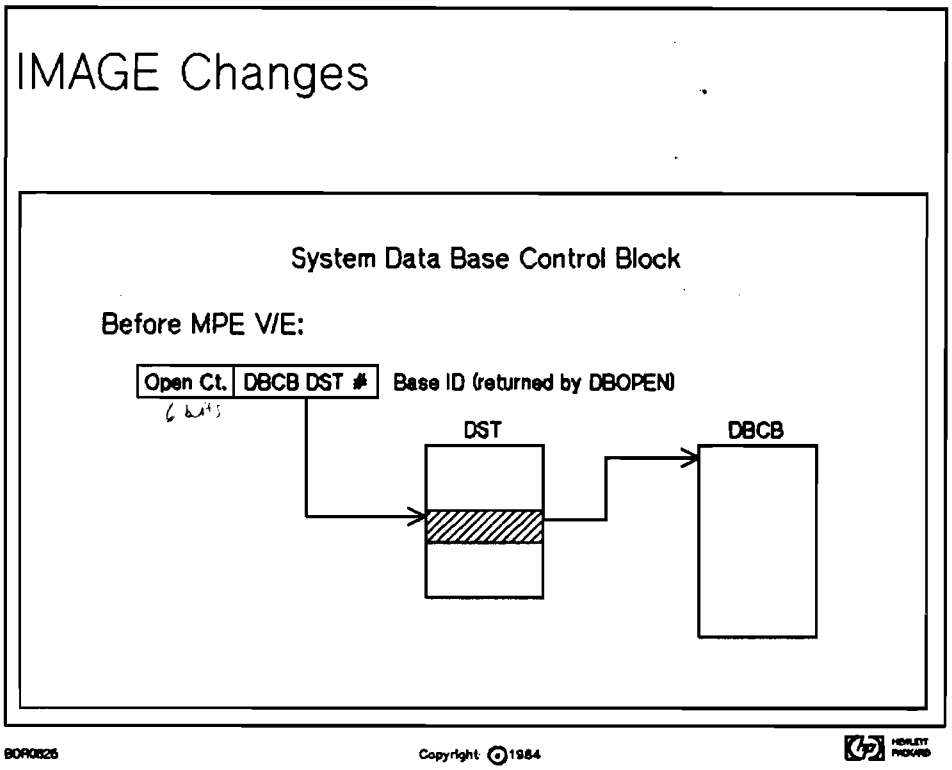
□ Notes

Effects of New Table Layouts

* Lock Area needed more space...
  So was expanded to 8K words (version B.04.51)
* DBCB has been expanded from 31K words to 32K words
  (version B.04.32)

□ MPE V/E

□ Notes

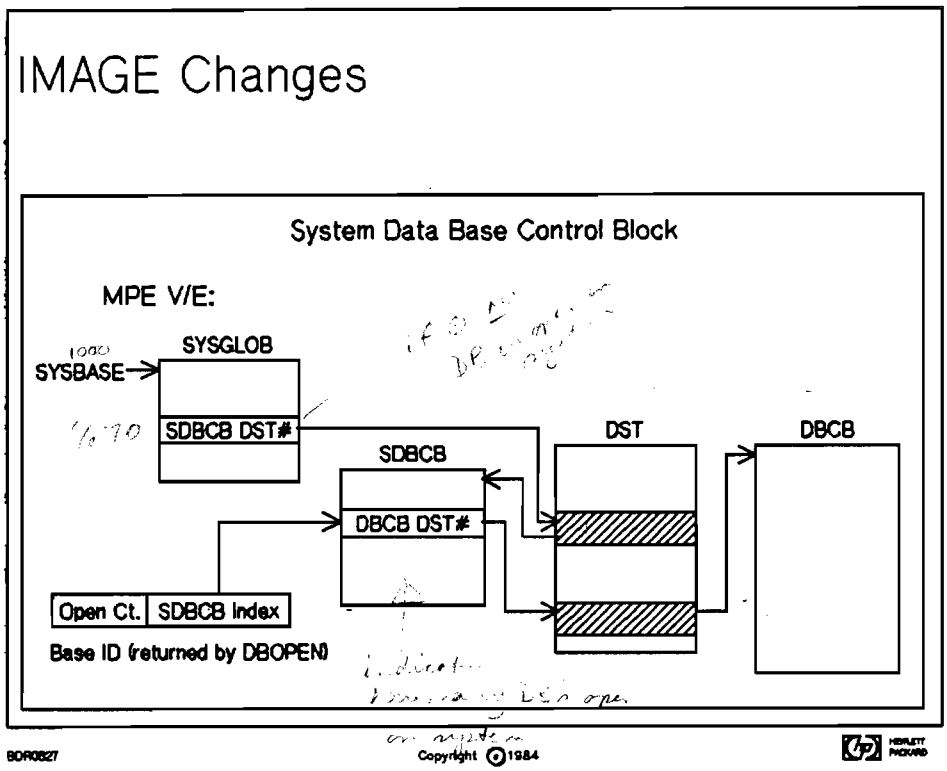


First 6 bits of BASEID is still the open count.

□ MPE V/E

□ Notes

IMAGE Changes



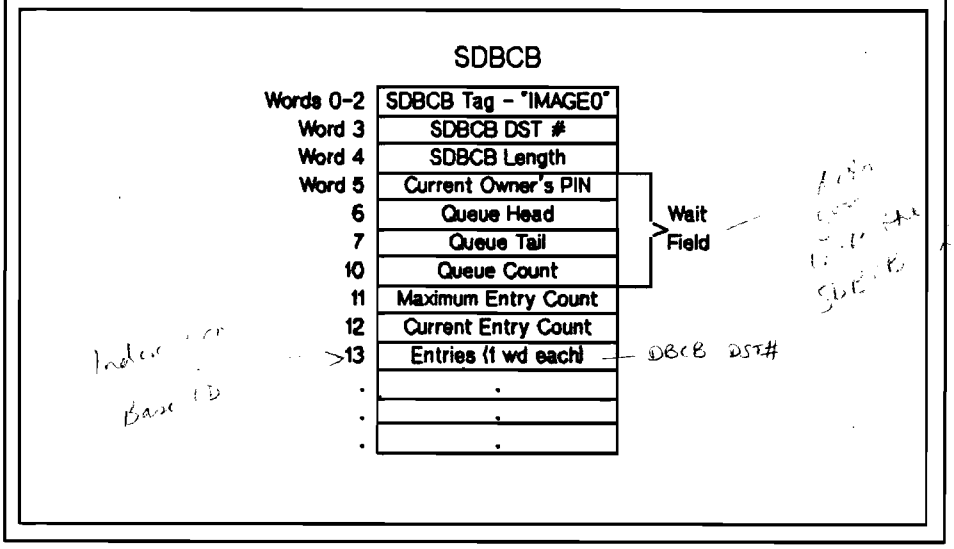
New level of indexing needed so that existing applications would not have to be modified.

Last 10 bits now contain index into SDCB.

□ MPE V/E

□ Notes

SDBCB Layout



Entries contain DBCB DST numbers.

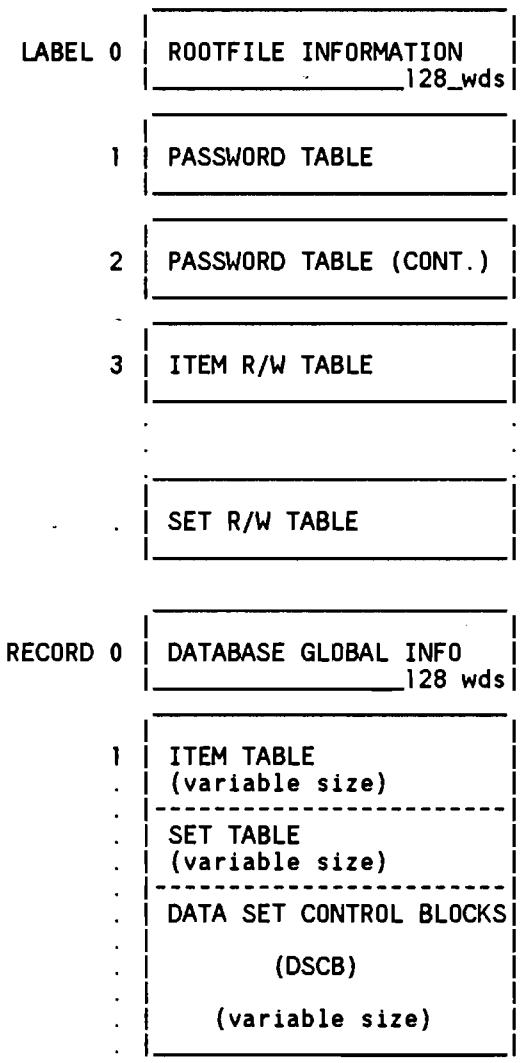
Wait flag used to gain control of SDBCB so no other modifications can take place (MPE OBTAIN).

Appendix A.1

IMAGE
Root File Layout

Appendix A.2

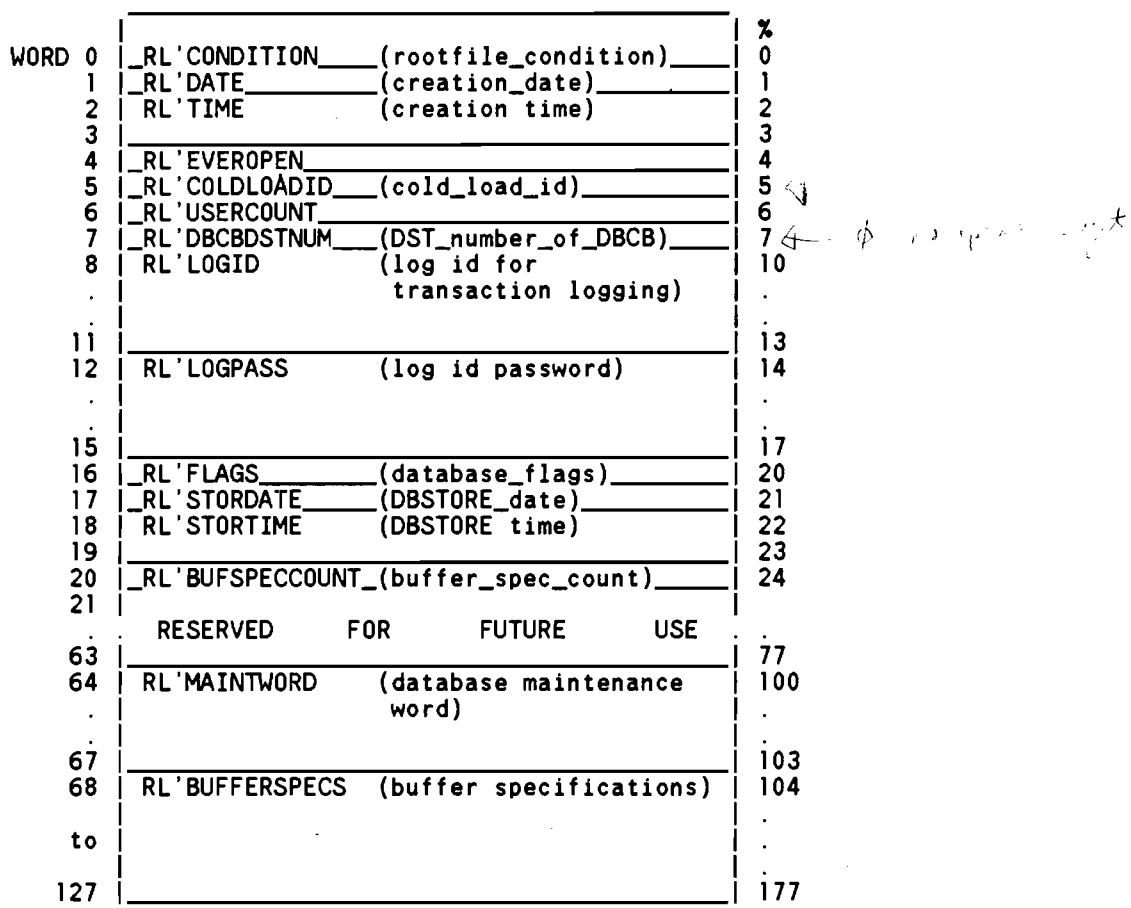
General Layout



The data base ROOT FILE is an MPE file with filecode equal to -400. The record size is 128 words, fixed, binary format with a blocking factor of 1. The size of the file depends on the number of data items and data sets defined in the data base.

Appendix A.3

Root File Label 0



RL'CONDITION (IN ASCII):

JB - Virgin. The database has not been created yet.
FW - OK. The database is OK.
RM - Modified deferred. The database is being modified.
MC - Maintenance create. The database is being created.
ME - Maintenance erase. The database is being erased.

Appendix A.4

Root File Label 0 (Continued)

RL'DATE: Root file creation date*. Its format is:

```
  0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15
|year_____|day_of_year_____|
```

RL'TIME: Root file creation time*. Its format is:

```
  0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15
|hour_____|minutes_____|
|seconds_____|tenth_of_seconds_____|
```

RL'EVEROPEN: This field is no longer used under IMAGE B

RL'FLAGS:

```
(0:1) - RECOVERY          Default is NO  (0)
(1:1) - LOGGING           Default is NO  (0)
(2:1) - ACCESS            Default is YES (1)
(3:1) - DUMPING           Default is NO  (0)
(4:1) - RESERVED-FOR-FUTURE-USE
(5:2) - SUBSYSTEM ACCESS Default is R/W  (00)
(7:3) - RESERVED-FOR-FUTURE-USE
(10:1)- DIRTY FLAG        Default is YES (1).
This indicates the data base has been modified but not restored.
(11:5)- RESERVED-FOR-FUTURE-USE
```

RL'STORDATE: Same format as RL'DATE*.

RL'STORTIME: Same format as RL'TIME*.

RL'BUFSPECCOUNT: Maximum number of buffer specifications allowed.

RL'MAINTWORD: For data bases with no maintenance word this field has 2 semicolons (';;') and trailing blanks.

RL'BUFFSPECS:

```
BIT/    0:_1:_2:_3:_4:_5:_6:_7:_8:_9:10:11:12:13:14:15   %
WD 68 |buffers_for_1 user_____|buffers_for_2 users_____| 104
   69 |buffers_for_3 users_____|buffers_for_4 users_____| 105
    . | etc...                                          |  .
    . |_________________________________________________|  .
  127 |buffers_for_119 users__|buffers_for_120 users__| 177
```

* The DATE and TIME fields can be formatted (for display purposes) individually by calling the FMTCALENDAR and FMTCLOCK Intrinsics respectivelly. Or both fields can be formatted at once with FMTDATE Intrinsic.

Appendix A.5

Root File Labels 1 & 2

LABEL #1

WORD			%
	0	Password for user class 0	0
	1	(this is a dummy field since user	1
	2	class 0 is not defined)	2
	3		3
	4	Password for user class 1	4
	5		5
	6		6
	7		7
	8	Password for user class 2	10
	9		11
	10		12
	11		13
	.		.
	.		.
	.		.
	124	Password for user class 31	174
	125		175
	126		176
	127		177

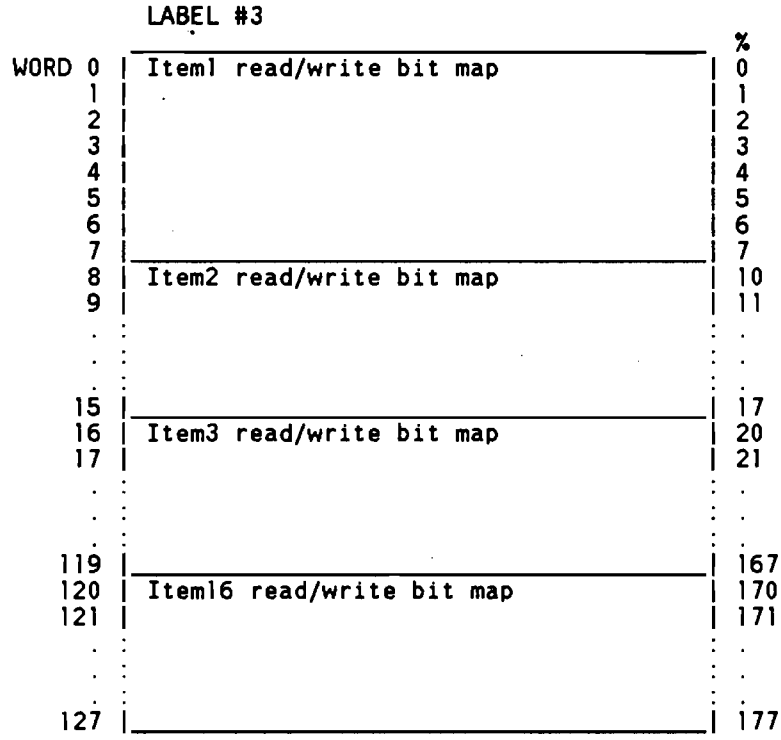
LABEL #2

0	Password for user class 32	0
1		1
2		2
3		3
4	Password for user class 33	4
5		5
6		6
7		7
8	Password for user class 34	10
9		11
10		12
11		13
.		.
.		.
.		.
124	Password for user class 63	174
125		175
126		176
127		177

The PASSWORD TABLE occupies user labels number 1 and 2. There are four words (8 characters) reserved for each password. The relative position of a password corresponds to the user class number defined in the schema. For user class numbers not defined in the SCHEMA, the four word field is filled with blanks.

Appendix A.6

Root File Label 3



The ITEM READ/WRITE TABLE starts in user label #3

There are eight words for each ITEM READ/WRITE bit map. For databases with more than 16 items, the read/write table continues in the next user labels. The specific format of this table is explained after the SET READ/WRITE TABLE since it is defined the same way. The number of user labels occupied by the ITEM READ/WRITE TABLE depends on the number of data items defined in the schema and can be obtained by rounding upwards (ceiling)the result of:

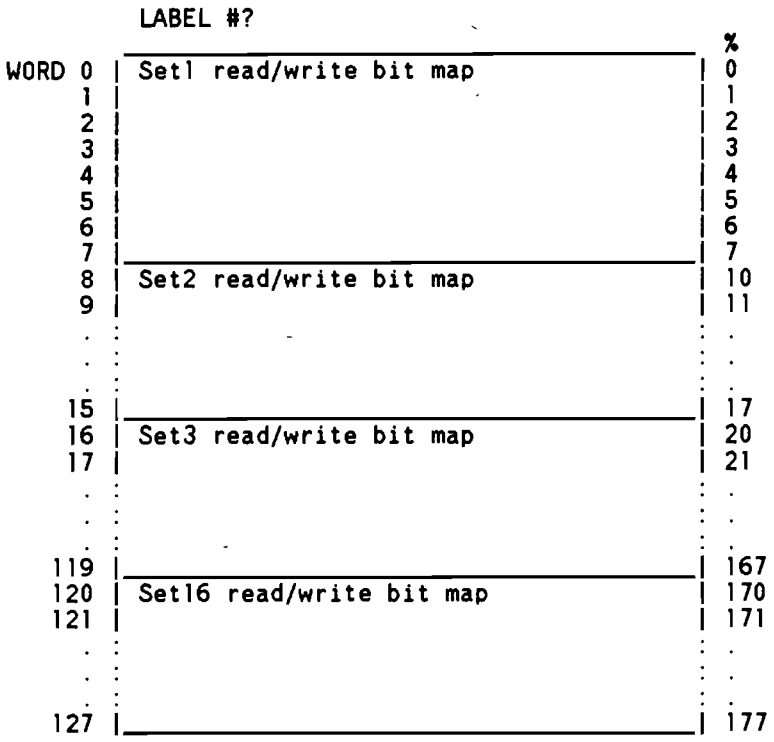
```
Num-of-labels = [(Num-of-items)*8]/128
```

Since there can only be a maximum of 255 data items in the schema, the maximum size for this table in user labels would be:

```
Max-size = [(255)*8]/128 = 15.93 => 16 labels.
```

Appendix A.7

Root File - Next Label



The SET READ/WRITE TABLE starts on a user label boundary after the ITEM READ/WRITE TABLE. There are eight words for each SET READ/WRITE bit map. For databases with more than 16 data sets, the read/write table continues in the next user labels. The specific format of this table is shown in the next page.

The number of user labels occupied by the SET READ/WRITE TABLE depends on the number of data sets defined in the schema, and is obtained by rounding upwards (ceiling) the result of:

```
Num-of-labels = [(Num-of-sets)*8]/128
```

Since there can only be a maximum of 99 data sets defined in the schema the maximum size for this table in user labels is:

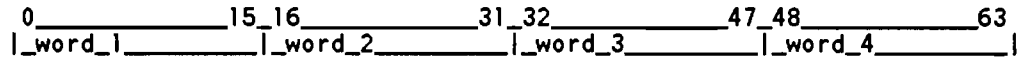
```
Max-size = [(99)*8]/128 = 6.18 => 7 labels
```

Appendix A.8

Item/Set Read/Write Table Format

There are eight words per item/set read/write table definition and up to 16 items/sets per record (user label). Within each 8 words, the first 4 words are the flags for the user classes which have read access to the item/set. The second 4 words are the flags for the user classes which have write access to the item/set. The detail format for an eight word field is shown below.

A. Four words for read access:

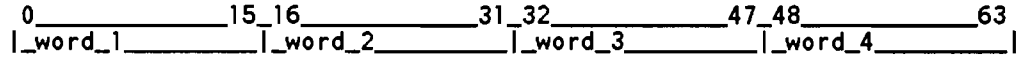


4 words represent 64 bits. Bit n represents read access for user class n to the item/set. If bit n is set to 1 then user class n has read access to the item/set. For example, if the word settings are:

word 1	word 2	word 3	word 4
%000016	%020000	%000410	%001300

This means that user classes 12, 13, 14, 18, 39, 44, 54, 56 and 57 have read access to the item/set. If no read/write security is defined at all for the item/set, then all of the read security bits are set to 1.

B. Four words for write access:



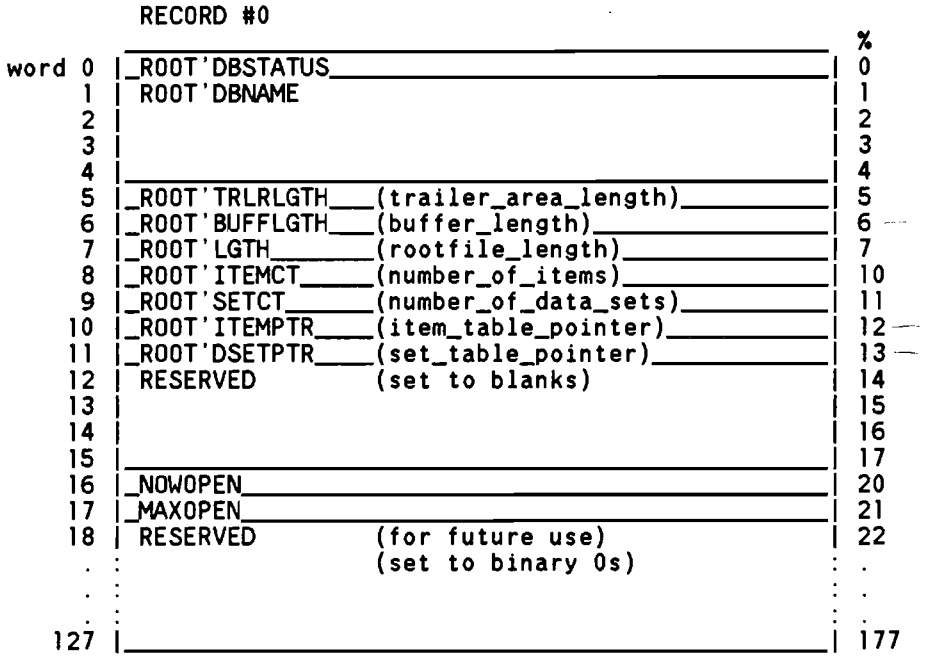
Write access flags have the same format as the read access flags.
Bit n represents write access for user class n to the item/set. If
bit n is set to 1, then user class n has write access to the item/set
For example, if the word settings are:

word 1	word 2	word 3	word 4
%000010	%020000	%000000	%001100

This means that the user classes 12, 18, 54 and 57 have write access to the item/set. If no read/write security is defined at all for the item/set, then all of the write security bits are set to 0.

Appendix A.9

Root File Record 0



ROOT'DBSTATUS:

(0:8) - IMAGE version ('B' in ASCII)
(8:8) - Binary 1 (filler)

ROOT'DBNAME: DATABASE name left justified (last 2 characters are blank).

NOWOPEN: Number of data sets opened. This field is not used in IMAGE B

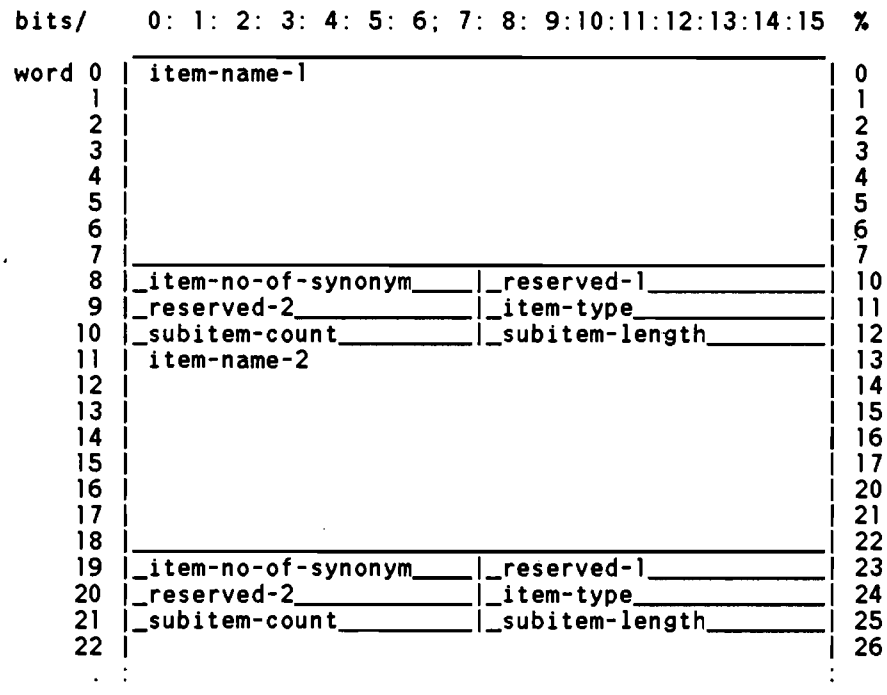
MAXOPEN: Maximum number of data sets that can be opened. This field is not used in IMAGE B.

NOTE: ROOT'ITEMPTR and ROOT'DSETPTR is a word offset from record 0 (beginning of the file, not including the space taken by the user labels) and can span several records.

These pointers point to the 0th entry of the table and since the 0th entry in the item table or the set table does not really exist, they actually point to 11 words before the beginning of the table. To get to the first entry in the table, this pointer should be incremented by the length of the entry (which is currently 11 words).

Appendix A.10

Root File Record 1



The ITEM TABLE starts in record #1.

Each entry is 11 words long and the length of the table depends on the number of data items defined in the schema. The relative position of an item definition depends on its relative position in the schema.

Item-name: is a data item name, left-justified and with trailing blanks.

Item-number-of-synonym: is the number of the item whose name has the same hashed result as this one (this is utilized for quick item name searches).

Item-type: is one of the following: I, J, K, R, X, U, Z, or P

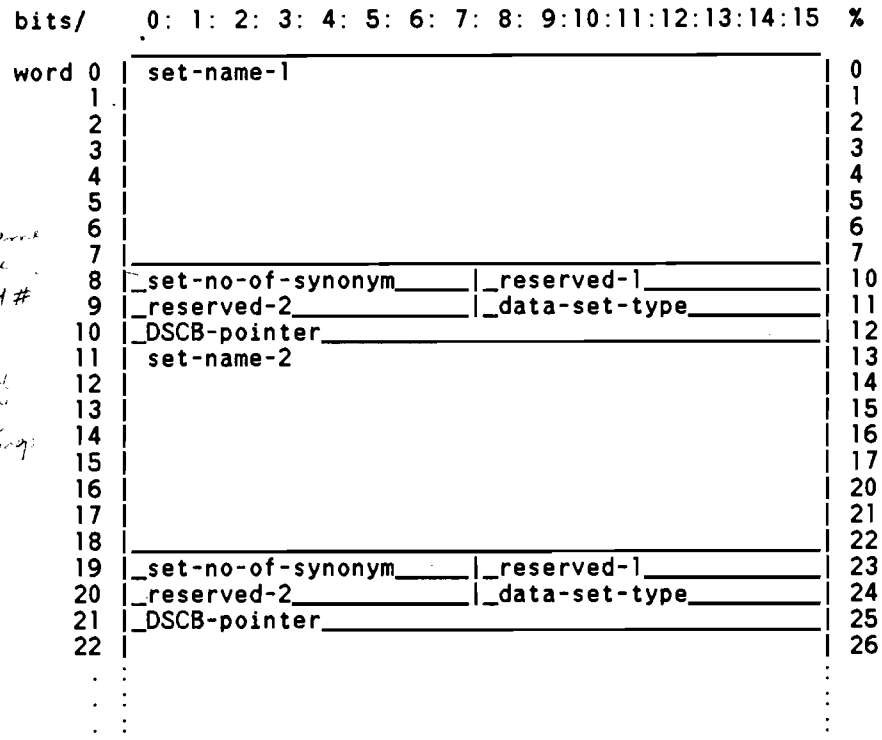
```
                item-type
                |
VALUES, 20J2;
          | |subitem-length
          |subitem-count
```

The maximum size for this table is 11*255 = 2805wds.

NOTES: The reserved-1 and reserved-2 fields are the 'old' level numbers for read and write security. Now, the values are always zero.

Appendix A.11

Root File - Continued



Set table follows the Item table.

Each entry is 11 words long. The length of the table depends on the number of data sets defined in the schema. The relative position of a set definition depends on its relative position in the schema.

Set-name: is a data set name, left-justified and with trailing blanks.

Set-number-of-synonym: is the number of a data set whose name has the same hashed result as this one (this is utilized for quick set name searches).

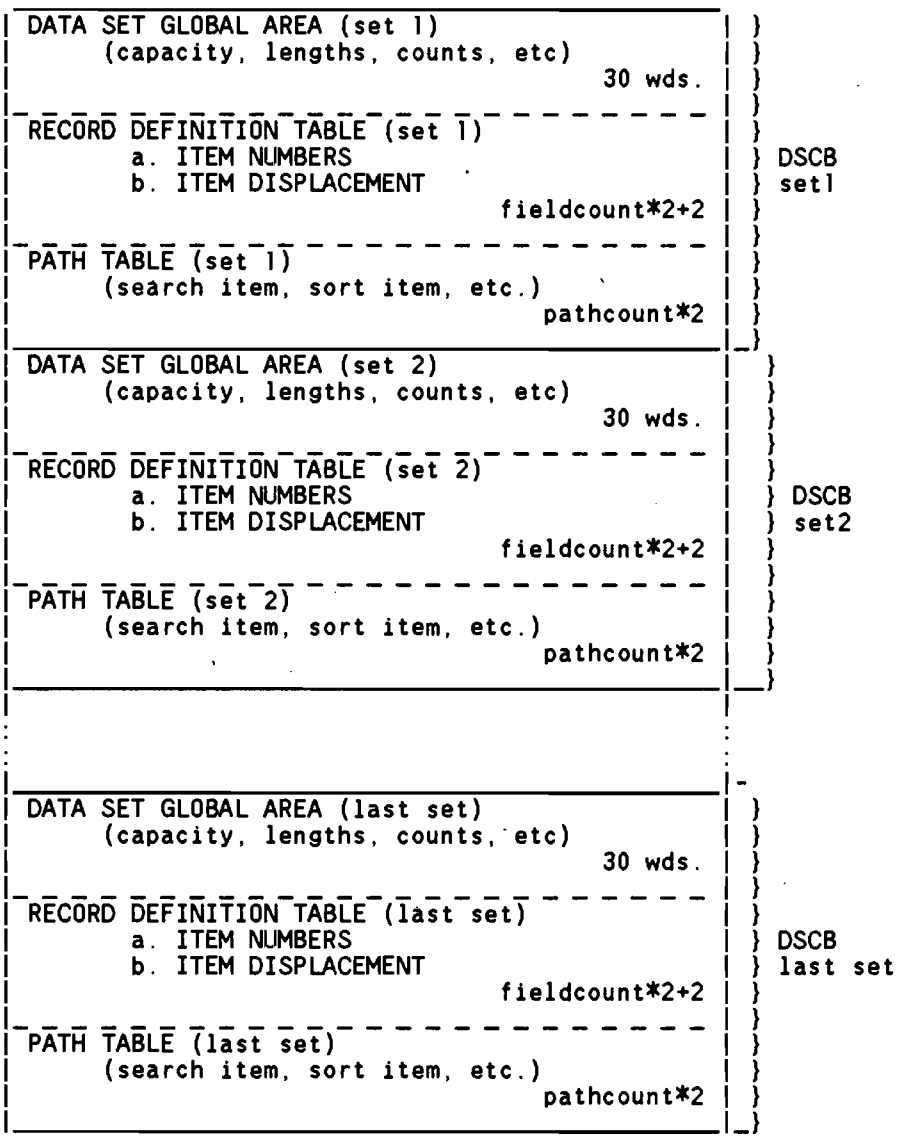
Data-set-type: is one of the following: A, M or D.

DSCB-pointer: is a pointer to the Data Set Control Block. This pointer is word offset from record #0. The DSCB is described ahead.

The maximum size for this table is 11*99 = 1089wds.

NOTES: The reserved-1 and reserved-2 fields are the 'old' level numbers for the read and write access respectively. Since this concept no longer applies, the values are set to zero.

Data Set Control Blocks (DSCB) – General Layout

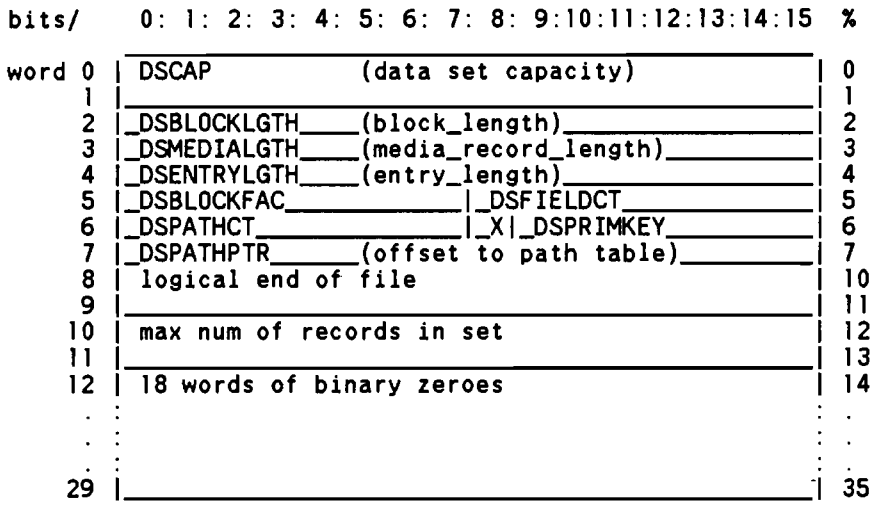


The DSCBs follow the SET TABLE in the Root File. There is one DSCB for each data set defined. The function of the DSCB is to define each data set within the data base.

Appendix A.13

Root File Next Record

DATA SET CONTROL BLOCK - GLOBAL AREA



DSCAP: data set capacity as reported by the SCHEMA processor.

DSBLOCKLGTH: data set block length including the bit map overhead.

DSMEDIALGTH: data set media record length (remember that this length includes the pointer overhead)

DSENTRYLGTH: data set entry length.

DSBLOCKFAC: data set blocking factor.

DSFIELDCT: data set field count. This is the number of fields specified for the data set.

DSPATHCT: data set path count. This is the number of paths that are specified for the data set.

X-DSKEYTYPE: data set key type. If DSKEYTYPE = TRUE then the key is hashed.

DSPRIMKEY: data set primary path or key. For master data sets, this is the field number of the search item. For detail data sets, this is the field number of the primary path.

DSPATHPTR: data set path table pointer. Word offset to the data set path table which contains an entry for each path defined. It points to path 0th entry in the table, so to get to the first entry the pointer should be incremented by the length of the entry (which is currently 2 words).

Appendix A.14

Root File - Next Record

DATA SET CONTROL BLOCK ITEM NUMBERS

```
          0: 1: 2: 3: 4: 5: 6: 7: 8: 9:10:11:12:13:14:15
         ___________________________________________________
word 0  |_item_num_of_1st_field_|_item_num_of_2nd_field_|
     1  |_item_num_of_3rd_field_|_etc._________________|
     .  |_etc._________________|_binary 0_____________|
     .  |_binary_0_____________|_binary_0_____________|
```

The Item Numbers Table follows the Global Area of the DSCB. The size of this table (in words) is equal to the number of items in the given data set plus 1. The first n bytes are used to carry the item numbers of the fields within the data set. The remaining n+2 bytes are set to binary zeroes.

Root File - Next Record

DATA SET CONTROL BLOCK
RECORD DEFINITION - ITEM DISPLACEMENT

```
          0: 1: 2: 3: 4: 5: 6: 7: 8: 9:10:11:12:13:14:15
         ___________________________________________________
word 0  |_word_offset_to_first_field___________________|
     1  |_word_offset_to_second_field__________________|
     2  |_word_offset_to_third_field___________________|
     .  :                                              :
     .  :                                              :
     .  :                                              :
     .  |_word_offset_to_last_field____________________|
        |_length_of_entry_____________________________|
```

This table immediately follows the Item Numbers Table.

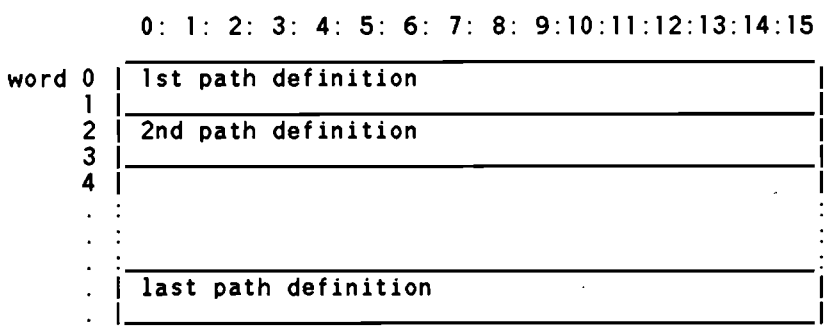
The word offset points to the starting location of the field within the media record. Remember that the media record includes the pointer over- head so this offset varies for master and detail data sets: if a master data set has only one path, the word offset for the first field is 10, since there are 10 words of overhead--5 words for the synonym chain pointers and 5 words for the data set chain head that it would be poin- ting to. On a detail data set with one path, the overhead is only 4 words.

The 'length-of-entry' field is the same as the media record length.

Appendix A.15

Root File - Next Record

DATA SET CONTROL BLOCK - PATH TABLE



There are 2 words (4 bytes) for each path definition. The PATH TABLE for master data sets has a different layout from the PATH TABLE for detail data sets.

Master sets:

Byte Description

1 - item number of the search item in the related detail set.

2 - item number of the sort item in the related detail set.

3 - set number of the related detail data set

4 - path number of the corresponding path in the related detail data set.

Detail sets:

Byte Description

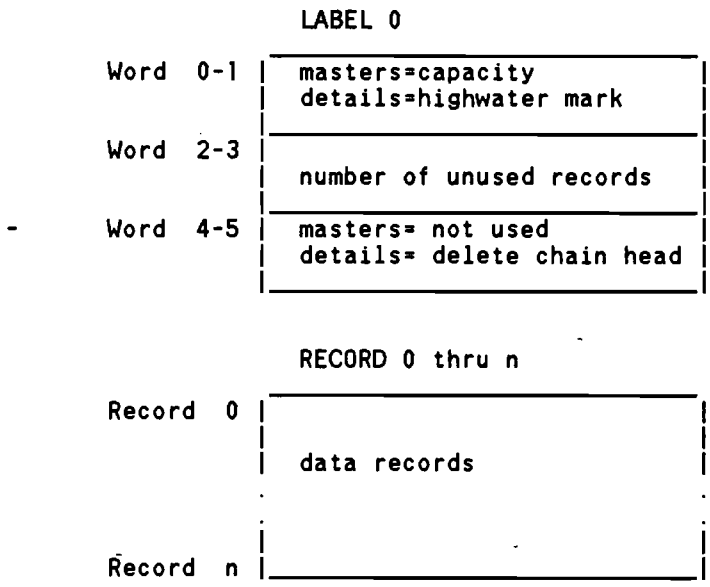
1 - field number of the search item.

2 - field number of the sort item.

3 - set number of the related master data set

4 - path number of the corresponding path in the related master data set.

General Data Set Layout



Data Set Label 0

Word 0-1: Record name of the highest readable record. For Masters, this is the highest record in the set (i.e. Capacity). For Details, this is the greatest number of records that have been written to the set thus far. For example, if there is room in the Detail data set for 100 records and 75 were written last week when the data set was loaded with DBLOAD, and yesterday 15 records were deleted from the data set, the High Water Mark is equal to a value of '75'.

Word 1-2: Number of unused records in the data set. This field is incremented when a record is deleted and decremented when a record is added. To determine the current number of entries used in the set subtract Word 1-2 (unused count) from Word 0-1 (capacity).

Word 3-4: The delete chain head for Details. This points to the record most recently deleted or contains a value of zero if no records have been deleted. This field is not used in Master data sets.

Appendix A.17

Data Set Records

The data in the data set records is arranged according to the Media records. These are formatted by the Schema Processor (DBSCHEMA).

Appendix B.2

Synonyms

a) use prime capacity
b) use byte keys
c) keep master 75% full

Buffer Specifications

a) set to a constant for wide range of users to avoid DBCB expansions and contractions
b) use more buffers for modifying
c) use output-deferred in stand-alone environment (DBCONTROL)
d) use formula to determine maximum number of buffers for DBCB

through DBUTIL

Blocksize

a) if larger blocksize, will have larger blocking factor
   i) fewer I/Os for sequential or chained reads
   ii) takes less time to chase/place synonyms
b) use larger blocksize if record is large

Sorted Chains

a) if not using extended (multi-level) sort, place sort item at end of entry
b) keep chains short
c) pre-sort before adding
d) changing sort item requires delete & put

Appendix B.3

List Parameter

a) partial
   i) names hashed to get item numbers
   ii) security checks done
   iii) data formatted in DBCB trailer, then passed to user's stack
   iv) use if few items needed from entry
b) "@"
   i) no hashing of names (numbers found in DBCB Data Set Control Block)
   ii) security checks done
   iii) data transferred directly to stack
c) "*"
   i) no look-up of numbers
   ii) no security checks on reads
   iii) if Current List = "@" then data transferred directly to stack
d) null or blank list
   i) only points to record; no data transferred
   ii) checks bit map to ensure record exists

DBUNLOAD/DBLOAD

a) use DBUNLOAD,CHAINED to reorganize chains and improve performance (fewer I/Os)
b) DBLOAD runs faster if buffer specifications are set high (50(1/2))

Paths

a) DBFIND changes path (Current Record in detail = 0), use multiple DBOPENs to maintain paths
b) better performance with fewer masters linked to one detail
c) make primary path the one most used

Programming Tips

a) keep code segments to around 4K words
b) keep code localized (try to keep procedures that call each other in the same segment)
c) use stacks efficiently (re-use data areas)
d) use item/set numbers instead of names
e) close data set when not needed
f) use DBGET Mode 1, if serially reading a master and deleting all entries
g) use DBCLOSE before serial reads
h) don't ignore errors

Appendix B.4

Schema

a) use set level write access
b) use dummy set/items to allow for future changes
c) automatic masters easier to use but have less control
d) use stand-alone masters for look-up tables
e) use stand-alone details to benefit from IMAGE security, Query, procedures, logging
f) consider blocksize
g) put items most used first in schema (faster conversion of names to numbers)

Query

a) debugging aid
b) not for heavy updating (no data edits)
c) QSLIST for restructuring
   i) delete sets
   ii) change item data types
   iii) change item length

Shared Access/Locking

a) lock around a transaction
b) keep lock as short as possible
c) set level locks okay for <= 8 IMAGE intrinsics
d) if using MR, lock in predefined sequence and unlock in reverse order
e) lock at same level, use same lock item
f) use DBGET Mode 1 in a lock after operator action
g) for serial or directed read, lock set/base
h) for chained or calculated read, lock entries
i) when changing lock item, must lock on both values
j) no need to lock in DBOPEN modes 3, 7, 8

Miscellaneous

a) consider contributed programs
b) consider data types and language used
c) can defer somes updates for batch
d) spread most volatile data sets across disc devices

Appendix B.1

IMAGE
Performance/Design Considerations