SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Data Base Management System (DMS) System Support Functions

User Guide/Programmer Reference

UP-8272 Rev. 3–B

This Library Memo announces the release and availability of Updating Package B to "SPERRY UNIVAC Operating System/3 (OS/3) Data Base Management System (DMS) System Support Functions User Guide/ Programmer Reference", UP-8272 Rev. 3.

This update describes the following features for release 8.0:

- Suppress subschema check
- Restore the data base with a dump file created from another data base
- Suppress quick-before-looks processing

Other changes include expanded descriptions, clarifications, or corrections that apply to the software before the current release.

Copies of Updating Package B are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisioned by your local Sperry Univac representative. To receive only the updating package, order UP-8272 Rev. 3–B. To receive the complete manual, order UP-8272 Rev. 3.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A09, B00, B09, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76, and 76U (Package B to UP-8272 Rev. 3, 53 pages plus Memo)	Library Memo for UP-8272 Rev. 3–B
		RELEASE DATE: September, 1982

PAGE STATUS SUMMARY

ISSUE: Update B — UP-8272 Rev. 3 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	B
Acknowledgment	1	Orig.
Preface	1	Orig.
Contents	1 2, 3 4 5	Orig. B Orig. B
1	1 thru 20	Orig.
2	1 thru 10 10a 11 12 thru 16 16a 17 18 19 20, 21 22 22a 23 24 25, 26 27, 28 28a 29 30 31, 32 33 34 35, 36 37 thru 41 42	Orig. B B Orig. A A Orig. A Orig. A A Orig. A Orig. B B A B Orig. B Orig. A Orig. A
3	1 thru 4 5 6, 7 8 8a 9 10 11, 12 12a 13, 14 15 16, 17	Orig. B Orig. B B* Orig. A B B* B Orig. B
4	1 thru 7 8 8a 9	Orig. A A Orig.
4 (cont)	10 11 thru 25 26 27 thru 32 33 34 34a 34b 35 thru 41	B Orig. B Orig. B Orig. B B* Orig.
5	1, 2 2a 3, 4 4a 5 6 7 8 thru 19	Orig. Orig. Orig. Orig. Orig. A B Orig.
Appendix A	1 thru 3 4 5, 6 7 thru 9 10 thru 14	B Orig. B A Orig.
Appendix B	1 thru 3	Orig.
Index	1 2 3 4 thru 6	A B A B
User Comment Sheet		

*New pages

All the technical changes are denoted by an arrow (➔) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (➔) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Contents

TABLES

The data base initializer (DBINT) is an offline utility processor that initializes pages within a data base. Page initialization includes building a page header and trailer into main storage, clearing the remainder of the page, and writing it to the data base mass storage area. DBINT does not determine whether the specified page contains any pertinent data before overwriting it. Therefore, DBINT should not be executed during a DBMS session; if it is, it should not be run against any data base files assigned to the DBMS. DBINT allows initialization of an area, a page range, or the entire data base.

DBINT must be executed twice during system generation: once to initialize the data dictionary data base, and once to initialize the user data base. The data dictionary data base must be initialized before executing any of the DMS language processors that load the data dictionary; the user data base must be initialized before executing any user applications that load the user data base.

If DBINT is executed following a system crash, the quick-before-looks file or files must be reinitialized (using // SCR or // LFD filename,,INIT). Otherwise, the next time the DBMS is activated, page images will be erroneously restored into the newly initialized data base.

NOTE:

The DBINT utility formats the data base files to an initial or empty state. Indiscriminate use of the utility may result in a compromised data base.

Figure 2–4 identifies the following functions performed by DBINT:

1. DUPL commands control DBINT.
2. DBINT loads the proper DMCL tables from the DBA library. (Note that both the DBINT load modules and the DMCL tables should reside in the same library.)
3. DBINT selectively initializes the data base (user or data dictionary).
4. DBINT issues output diagnostic and historical information.

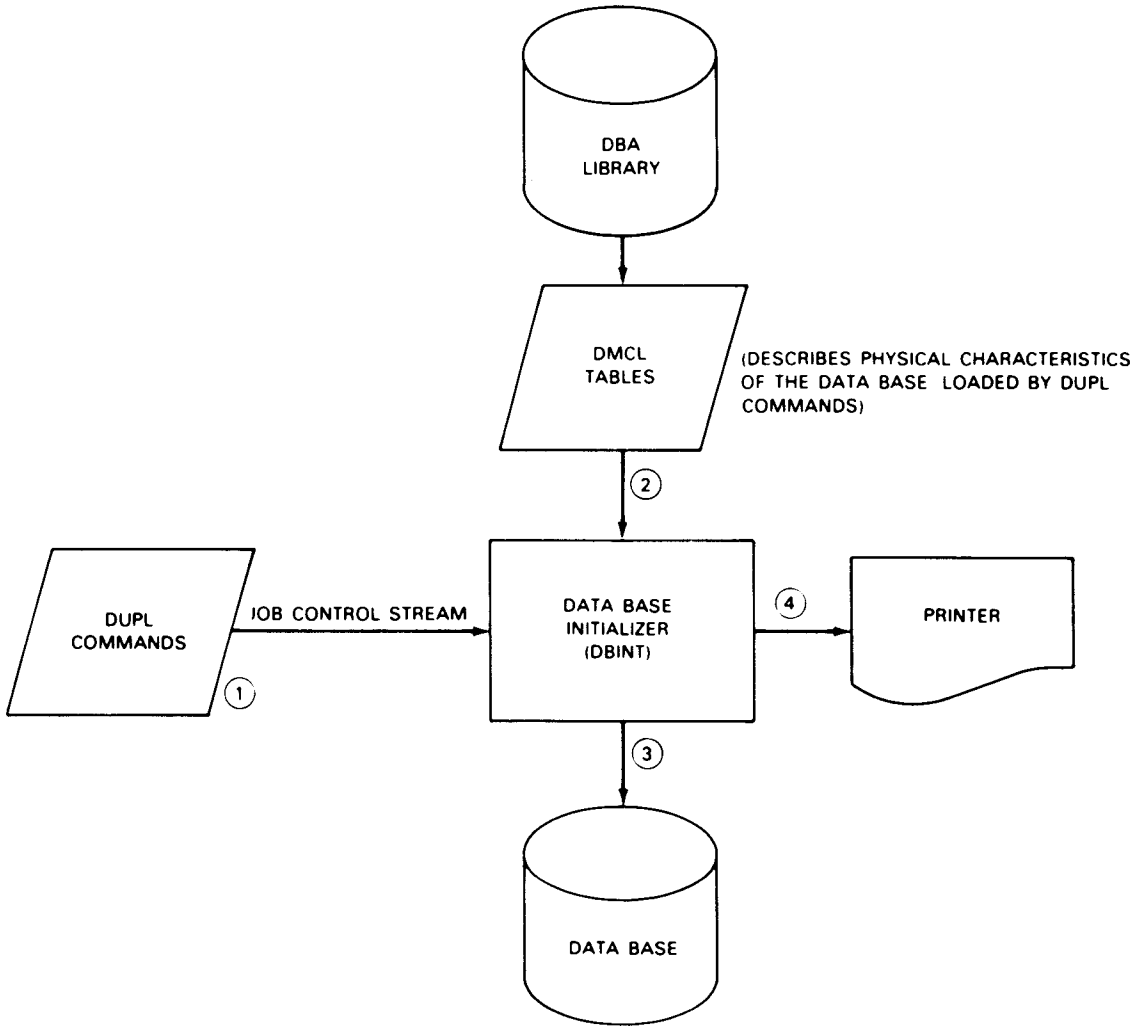


Figure 2—4. Data Base Initialization

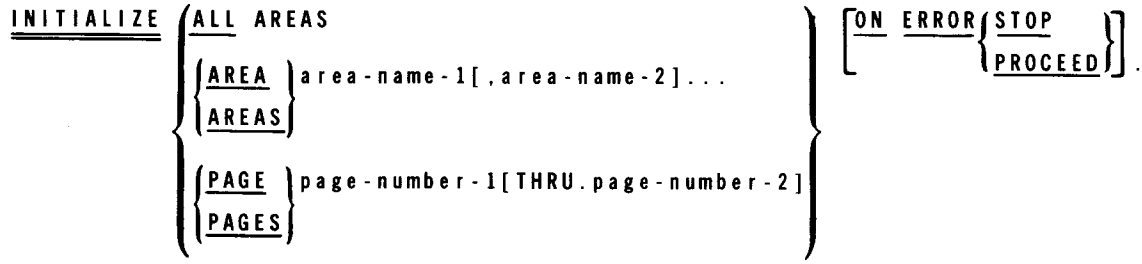
The first command acceptable to DBINT is the DMCL command. After DBINT successfully loads the DMCL tables (Figure 2-4), it attempts to allocate storage for:

- File access and DTFs
- Data base page buffer
- Space inventory page buffer

Sufficient memory must be specified on the // JOB statement. Failure to allocate storage for any of these generates an error message and terminates the utility.

All data base files which have access-control keys must be identified to DBINT with DATA BASE FILE commands. The ACTIVATE DMCL command must always be issued. DBINT may then process INITIALIZE commands until it receives an END command. As each data base page is initialized, the proper space inventory record is modified to indicate the maximum space available. The INITIALIZE command specifies the areas or pages to be initialized on the data base. Its format follows:

INITIALIZE Command Format:



where:

`area-name-1 area-name-2`
Specifies the area names defined in the area section of the schema data division in the DDL description.

`page-number-1 page-number-2`
Specifies an integer value for page number.

Syntax Rules:

1. The page-number-1 value must be less than or equal to page-number-2.
2. If the THRU clause is omitted, only one page will be initialized.
3. The page range may include any set of contiguous pages allocated to the data base (extendable pages are not allowed).
4. The ALL option initializes the data base by successive areas.
5. The ON ERROR clause operates as described in 1.4.

PRNTR-N
The job control procedure is not to generate a printer device assignment set. This allows the insertion of LCB and VFB job control statements.

SCHKEY-schema-unlock-key
The 1- to 4-character literal identifying the schema key that authorizes access to the schema network.

NOTE:

If DMCL compilation terminates with DMS status information being printed or the system terminates abnormally, the data dictionary data base must be recovered (2.3.3.1) before you can rerun the compilation.

The DMCLP call to the DMS jproc executes the DMCLP in standard mode, then assembles and links the DMCL for the user data base. If an error occurs during DMCL processing, the DMCLP sets the UPSI byte. (See the data description language user guide/programmer reference, UP-8022 (current version).) The DMCLP call tests the UPSI byte and bypasses the assembly and linkage steps if it finds a fatal error (X'80'). For this reason, you may want to reset UPSI to zero before calling the jproc.

The following sample job control stream uses the DMCLP call to the DMS jproc to create the DMCL for your data base.

```
1                                                                    72
// JOB DBDMCL
// WRTBIG '    DATABASE    ','    DMCL    ','COMPILATION','DAT$','TIM$'
// DISKØ1 JSET    'DISKØ1'
// DMCLP IN (&DISKØ1.DBA:SOURCE),INMOD SPDMCL,OUTMOD SPDMCL,              X
//1  OUT (&DISKØ1.DBA.LIBRARY),DMCL DMCLDD, ALTLOD (&DISKØ1.DBA.LIBRARY)
/&
```

The following sample job control stream initializes your data base using DUPL syntax. For details of the use of the DBINT utility, its DUPL commands, and the DBINT call to the DMS jproc, see 2.3.3.

```
// JOB SPINITDB...CØØØ
// DVC 2Ø  // LFD PRNTR
// DISKØ1 JSET  'DMS9ØX'
// DVCVOL &DISKØ1
// EXT ST.C.,BLK.(2Ø48,15)
// LBL 'SAMPLE DATABASE CUST'  // LFD DBMFIL1
// DVCVOL &DISKØ1
```

(continued)

```
// EXT ST.C..BLK.(2048.15)
// LBL 'SAMPLE DATABASE PRODORD'  // LFD DBMFIL2
// DVCVOL &DISK01
// LBL 'DBA LIBRARY' // LFD LOAD
// EXEC DBINT.LOAD
/$
DMCL IS SPDMCL.
DATA BASE IS DBMFIL1 ACCESS-CONTROL KEY IS 'CUST'.
DATA BASE IS DBMFIL2 ACCESS-CONTROL KEY IS 'PROD'.
ACTIVATE DMCL.
INITIALIZE ALL AREAS.
END.
/*
/&
```

2.3.6. Subschema Creation and Compilation

The data dictionary contains only one schema network; however, many subschema networks can be in the data dictionary. Each subschema network placed in the data dictionary requires the following steps:

1. Creation of a source file containing the subschema source code. (See data description language programmer reference, UP-8022 (current version).)

2. Start-up of the DBMS job if not already active. (See 2.3.4.)

3. Compilation of the source subschema using SUBSP.

You can create and compile the subschema any time after the schema is processed and you can re-create a subschema any number of times. Whenever you recompile a subschema, you should also recompile all the application programs that invoke the subschema. In this way, you can be sure they reflect the new record descriptions from the recompiled subschema. Since inconsistencies such as differing record lengths could cause DML verb processing to overlay record images in working storage, DMS guards against possible contamination of the data base. It does so by terminating programs whose subschema date and time do not match those of the current subschema. You can execute an application program with a date and time that do not match those of the current subschema by including the SUPPRESS SUB-SCHEMA CHECK statement at DBMS start-up (3.3.1.1.6).

You can compile subschemas using DUPL commands or the SUBSP call to the DMS jproc to operate the SUBSP processor. The following DUPL commands are valid for executing the subschema processor either for initial subschema processing or for changing an already existing subschema.

```
DMCL FOR DATA DICTIONARY IS dmcl-name.

CHANGE {LOCK  } FOR SUB-SCHEMA subschema-name OF SCHEMA schema-name
       {LOCKS }

    [ [ LOCK FOR ALTER IS literal-1                          ] ]...
    [ [ LOCK FOR COPY IS literal-2                           ] ]
    [ [ KEY FOR ALTER OF SUB-SCHEMA IS literal-3             ] ]
    [ [ KEY FOR COPY OF SCHEMA IS literal-4                  ] ]

COMPILE SUB-SCHEMA [USE SOURCE FILE lfd-name MODULE module-name]
    [USE OUTPUT FILE lfd-name MODULE module-name]

    [ [ LOCK FOR ALTER IS literal-1                          ] ]...
    [ [ LOCK FOR COPY IS literal-2                           ] ]
    [ [ KEY FOR ALTER OF SUB-SCHEMA IS literal-3             ] ]
    [ [ KEY FOR COPY OF SCHEMA IS literal-4                  ] ]

END PROCESSING.
```

Syntax Rules:

1. The DMCL statement must precede the other statements in the command stream.

2. The dmcl-name and subschema-names are 1- to 6-character words; the schema name is a 1- to 8-character word. They must correspond exactly with the names in the data dictionary (or in the DBA library).

3. SOURCE FILE lfd-name is a 1- to 7-character literal giving the lfd-name for the file containing the source. Module-name is a 1- to 8-character word specifying the name of the source module.

4. OUTPUT FILE lfd-name is a 1- to 7-character word that specifies the file to contain the generated assembly language source module. The module name is a 1- to 8-character word.

5. Literal-n is 1 to 4 characters.

Processing Rules:

1. The COMPILE clause specifies compilation or recompilation of a subschema. The optional USE statement identifies the file containing the source to be compiled. If you omit this statement, the source is read from the command stream, following the COMPILE statement. If the source cards are in the command stream, the last card must be followed by a card containing *END in the first four positions.

2. The optional OUTPUT clause identifies the file to which the object output is generated. If omitted, a module with the name SBSOUT is generated to YRUN.

3. The LOCK FOR ALTER clause is optional. If present, the given lock value is inserted for the subschema. The ALTER lock prohibits recompilation of the subschema, unless you specify KEY FOR ALTER OF SUBSCHEMA.

4. The LOCK FOR COPY clause is optional. If present, the given lock value is inserted for this subschema. The COPY lock prohibits DML preprocessing unless you specify the KEY clause in your DUPL command stream.

5. The KEY FOR SCHEMA clause must be present if the schema has COPY lock specified. The value must correspond with the COPY lock value.

6. You must specify the KEY FOR SUB-SCHEMA clause if the subschema has an ALTER lock applied to it, and the value must correspond to the (previous) lock value.

7. The CHANGE clause specifies that ALTER and/or COPY locks are to be changed, without recompilation of the subschema. If the subschema already has an ALTER lock, you must specify the KEY clause in the command stream. An ALTER lock clause in the command stream imposes the specified value as a lock on the subschema for updating, and a COPY lock clause imposes the specified value as a lock on the subschema for retrieval. The absence of a lock clause in the command stream is interpreted as no change to existing locks. The word NULL specified for a lock removes the existing lock.

8. The KEY FOR SCHEMA clause must be present if the schema has a copy lock specified. The value must correspond with the COPY lock value.

9. The END PROCESSING statement terminates the processing.

NOTE:

If subschema compilation terminates with DMS status information being printed or the system terminates abnormally, the data dictionary data base must be recovered (2.3.3.1) before you can rerun the compilation.

A sample job control stream to compile, assemble, and link the subschema using DUPL commands follows:

```
// JOB SPSUBS...14000
// DVC 20  // LFD PRNTR
// DISK01 JSET  'DMS90X'
// DVCVOL &DISK01
// LBL 'DBA SOURCE'  // LFD SRCIN
// DVCVOL &DISK01
// LBL 'DBA LIBRARY' // LFD LOAD
// EXEC SUPSP.LOAD
/$
DMCL IS DMCLDD.
COMPILE SUB-SCHEMA
USE SOURCE FILE SRCIN MODULE SPSUBS
USE OUTPUT FILE $Y$RUN MODULE SPSUBS.
END.
/*
// WORK1
// WORK2
// EXEC ASM
// PARAM IN=SPSUBS/$Y$RUN
// WORK1
// EXEC LNKEDT
// PARAM OUT=LOAD
/$
         LOADM SPSUBS
         INCLUDE SPSUBS.$Y$RUN
/*
/&
```

PRNTR=[lu][,vol-ser-no]
Default logical unit number and optional volume serial number.

PRNTR=N
The job control procedure is not to generate a printer device assignment set. This allows the insertion of LCB and VFB job control statements.

SBSKEY=subschema-unlock-key
The 1- to 4-character literal that names the subschema key that authorizes access to the subschema network.

SCHKEY=schema-unlock-key
The 1- to 4-character literal that names the schema key that authorizes access to the schema network.

NOTE:

If subschema compilation terminates with DMS status information being printed or the system terminates abnormally, the data dictionary data base must be recovered (2.3.3.1) before you can rerun the compilation.

The SUBSP call to the DMS jproc executes the subschema compiler, then assembles and links the subschema. If an error occurs during subschema processing, the subschema processor sets the UPSI byte. (See the DMS data description language user guide/programmer reference, UP-8022 (current version).) The SUBSP call tests the UPSI byte and bypasses the assembly and linkage steps if it finds a fatal error (X'80'). For this reason, you may want to reset UPSI to zero before calling the jproc.

The following job control stream compiles, assembles, and links a sample subschema:

```
1                                                                    72
// JOB SUBSCHEM
// WRTBIG 'SUBSCHEM','COMPILATION','DAT$','TIM$'
// DISKØ1 JSET     'DISKØ1'
// SUBSP IN=(&DISKØ1.DBA.SOURCE),INMOD=SPSUBS,OUTMOD=SPSSUBS,          X
//1 OUT=(&DISKØ1.DBA.LIBRARY),DMCL=DMCLDD,ALTLOD=(&DISKØ1.DBA.LIBRARY)
/&
```

2.3.7. Application Program Creation, Linking, and Execution

Your COBOL programs with DML verbs must be preprocessed by the DML preprocessor (DMLP) before COBOL compilation. The processor inserts subschema and other information in the data division of the COBOL/DML programs and replaces all DML verbs with COBOL call statements that call the DBMS run-time modules. After DML preprocessing, you can compile and link the output from DMLP with the DBMS interface modules. DBMS interface modules are supplied in YOBJ.

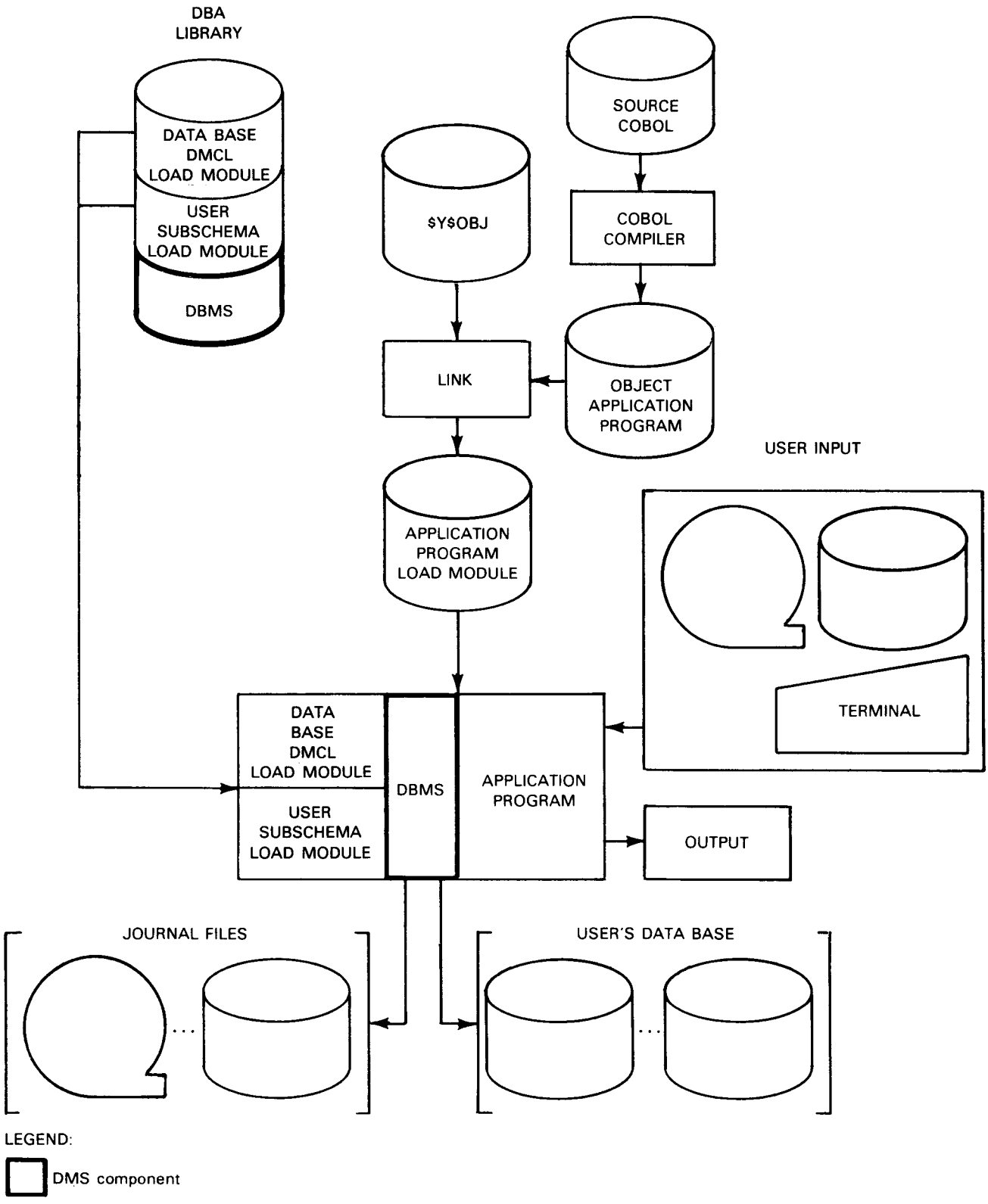


Figure 2—5. Compiling, Linking, and Executing Application Program

The DBA uses the following start-up syntax to direct DBMS start-up processing. At this time, the DBA specifies the number and type of run units that may be active during the session. For a detailed description of each DBMS start-up statement, see 3.3.1.1 and 3.3.1.2.

```
DBMS SECTION.
MAXIMUM NUMBER OF DBMS RUN-UNITS IS integer.

JOURNAL FILE IS {lfd-name} [ACCESS-CONTROL KEY IS literal].
               {DUMMY    }

[MAXIMUM NUMBER OF IMS-THREADS IS integer.]
[MAXIMUM NUMBER OF IMS-TERMINALS IS integer.]
[SUPPRESS SUB-SCHEMA DATE TIME CHECK.]
DMCL SECTION.
DMCL IS dmcl-name.
[ASSIGN integer DATA BASE BUFFERS.]
[ASSIGN integer SPACE INVENTORY BUFFERS.]
[ALLOCATE integer KEEP LOCKS.]
[MAXIMUM NUMBER OF RUN-UNITS IS integer.]
[MAXIMUM NUMBER OF UPDATING RUN-UNITS IS integer.]
[DATA BASE FILE IS lfd-name[ACCESS-CONTROL KEY IS literal].]
[QUICK-BEFORE-LOOKS FILE IS lfd-name[ACCESS-CONTROL KEY IS literal]
      [ON ERROR {STOP   }].                                        ]
                {PROCEED}

[IMS QUICK-BEFORE-LOOKS FILE IS lfd-name                           ]
          [ACCESS-CONTROL KEY IS literal]
          [ON ERROR {STOP   }].                                    ]
                    {PROCEED}

[CALC ROUTINE IS {CALC00}.]
                {CALC01}
ACTIVATE DMCL.
END.
```

DBMS start-up processing requires that a printer be available during the initial phase of establishing the DMS environment. The printer is acquired through the normal // DVC and // LFD job control statement sequence and is dynamically released upon completion of the start-up output. Two print files named PRNTR and PRNTX are required.

3.3.1.1. DBMS Section

The DBMS section is required to define the global attributes for a DBMS session. The syntax provided in this section as well as that in the DMCL section allows the DBA to control the DMS environment during a DBMS session. If the DBA wishes to change the DBMS environment, he must terminate the session, change the syntax, and start up another session. The following statements make up the DBMS section.

3.3.1.1.1. DBMS SECTION Statement

The DBMS SECTION statement is a required section definition statement.

Format:

```
DBMS SECTION.
```

3.3.1.1.2. MAXIMUM DBMS RUN-UNITS Statement

The MAXIMUM DBMS RUN-UNITS statement defines the maximum number of DMS run units that may be active during a DBMS session. (A run unit is a unit of work initiated by an IMPART verb and terminated by a DEPART verb.)

Format:

```
MAXIMUM NUMBER OF DBMS RUN-UNITS IS integer.
```

where:

integer
: Indicates an unsigned number with a valid range of 1 to 6. If the integer value is not within the specified range, start-up will set the integer to 6.

3.3.1.1.3. JOURNAL FILE Statement

The JOURNAL FILE statement names the journal file to be used for the duration of the session. DBMS writes area looks and checkpoints to this file only for DMS run units.

Format:

```
JOURNAL FILE IS { lfd-name } [ACCESS-CONTROL KEY IS literal].
               { DUMMY    }
```

where:

lfd-name
: Specifies the same 1- to 7-character alphanumeric filename (the first character must be alphabetic) that appears in the job's LFD statement and also in the DMCL source.

DUMMY
Specifies that journal processing is ignored. In this case, the DVC and LFD job control statement sequence for the journal file is not necessary.

literal
Specifies the ACCESS-CONTROL KEY with the same name as the LOCK that was optionally applied to the journal file in the DMCL source.

Rules:

1. As the DMCLs are loaded, both the lfd-name and the ACCESS-CONTROL KEY literal are validated with the same specifications in the DMCLs active for this session. If they do not compare, an error message is issued and start-up is terminated. Start-up is also terminated if there are any errors reported during the actual journal file open processing.

2. The journal file contains standard labels. It is advisable for the DBA to use the file-identifier field to uniquely identify journal files. Inclusion of a date/time stamp or sequence number in this field should be considered.

3. The user should consider password protecting his DMS files via the OS/3 CAT and LBL job control statements.

3.3.1.1.4. MAXIMUM IMS-THREADS Statement

The MAXIMUM IMS-THREADS statement indicates the maximum number of IMS subtasks that may access this DMCL concurrently.

Format:

```
[MAXIMUM NUMBER OF IMS-THREADS IS integer.]
```

where:

integer
Is an unsigned number greater than zero and equal to at least the minimum number of subtasks to be active. A value greater than the minimum results in better performance by increasing throughput.

Rules:

1. If the integer value is not greater than zero, DBMS defaults to 1.

2. When this statement is issued, the MAXIMUM IMS-TERMINALS statement must also be issued.

3. An integer value of 1 should be used for the IMS single-thread interface.

3.3.1.1.5. MAXIMUM IMS-TERMINALS Statement

The MAXIMUM IMS-TERMINALS statement defines the maximum number of terminals allowed online at one time.

Format:

```
[MAXIMUM NUMBER OF IMS-TERMINALS IS integer.]
```

where:

integer
: Is an unsigned number not exceeding the value assigned to the TERMS parameter in the IMS configurator NETWORK section. (Refer to IMS system support functions user guide/programmer reference, UP-8364 (current version) for further information.)

Rules:

1. If the integer value is not greater than 0, DBMS defaults to 1.
2. When you issue this statement, you must also issue the MAXIMUM IMS-THREADS statement.

3.3.1.1.6. SUPPRESS SUB-SCHEMA CHECK Statement

The SUPPRESS SUB-SCHEMA CHECK statement allows application programs to execute even though their subschema date and time fields do not match those of the current subschema.

Format:

```
[SUPPRESS SUB-SCHEMA DATE TIME CHECK.]
```

This statement should not be used for normal processing. It should be included only when it is necessary to inhibit the subschema date and time check.

The SUPPRESS statement bypasses normal comparison of the subschema date and time between subschema load modules and application programs that reference those subschemas. The statement remains in effect throughout a DBMS session.

Each subschema load module is marked with the date and time of compilation. Each application program that invokes the subschema is marked with the same date and time; the DML preprocessor includes this data in the DMCA. During IMPART or BIND processing, DMS checks that the date and time in the subschema are the same as the date and time in the application program. If they do not agree, DMS displays the error message:

QW39 SUB-SCHEMA COMPILATION DATE/TIME MISMATCH

and terminates the program. This prevents possible overlaying of record images in working storage and consequent contamination of the data base.

Normally, when a subschema has been recompiled, all its associated application programs are also recompiled and no date or time mismatches occur. Therefore, the SUPPRESS statement should not be used except in emergencies. For example, if a load library is lost, it must be re-created by recompiling the subschema. In this situation, the date and time fields will not match and the SUPPRESS statement may then be included temporarily in order to let data base processing proceed.

If an unforeseen date or time mismatch occurs, you should investigate the reason for the discrepancy. It is not advisable to suppress the date and time check until you are sure why it occurred.

3.3.1.2. DMCL Section

The DMCL section is required to specify all the DMCLs to be loaded for a given session. The DMCL and its associated statements are repeated for each DMCL to be loaded. However, only the actual DMCL and ACTIVATE statements are required. By using the optional statements associated with the DMCL statement, the DBA can override the DMCL specification for buffer space, declare additional space for KEEP locks, define the number and types of run units, validate lock-protected files, and select a CALC routine.

NOTE:

The data base page size for all DMCLs activated at a given DBMS session must be the same size because the journal file does not support variable length records. The data base page sizes are restricted to 8K when journaling to a UNISERVO VI-C tape drive that supports a maximum block size of 8K.

3.3.1.2.1. DMCL SECTION Statement

The DMCL SECTION statement is a required section definition statement.

Format:

```
DMCL SECTION.
```

where:

integer
Is an unsigned number with a valid range of 1 to 6.

Rules:

1. If the integer value is not within the specified range, a warning message is issued and the value defaults to the corresponding value given in the DMCL source.

2. If this statement is omitted, the value assigned is that specified in the DMCL source.

3.3.1.2.7. MAXIMUM UPDATING RUN-UNITS Statement

The MAXIMUM UPDATING RUN-UNITS statement indicates the maximum number of concurrently updating DMS run units that may access this DMCL. This statement is used to determine the number of partitions for the quick-before-looks file.

Format:

```
[MAXIMUM NUMBER OF UPDATING RUN-UNITS IS integer.]
```

where:

integer
Is an unsigned number with a valid range of 0 to 6.

Rules:

1. If the integer value is not within the specified range, a warning message is issued and the value defaults to the corresponding value given in the DMCL source.

2. If this statement is omitted, the value assigned is that specified in the DMCL source.

3. By specifying zero updating run units, you enforce read-only access for this DMCL. The following limitations apply:

   a. You must specify quick-before-looks for every area in which an update is attempted.

   b. You must not specify DUMMY for the quick-before-looks lfdname.

   Any attempt to update while zero updating run units is in effect will cause this error message to be displayed:

   QW22 MAXIMUM UPDATING RUN-UNITS EXCEEDED FOR DMCL

   The message may also appear if a DML program opens the data base for update, even before the program attempts an explicit update operation. The problem can occur if DBMS tries to complete a deferred deletion (that is, to physically purge records that were logically deleted at an earlier time).

3.3.1.2.8. DATA BASE Statement

DATA BASE statements are required for all lock-protected files to be opened for this session. You must repeat this statement for each file.

Format:

```
[DATA BASE FILE IS lfd-name[ACCESS-CONTROL KEY IS literal].]
```

where:

lfd-name
: Is the same 1- to 7-character alphanumeric file name (the first character is alphabetic) that appears in the job's LFD job control statement and also in the DMCL source.

literal
: Indicates the same name as the LOCK that was optionally applied to the data base file in the DMCL source.

Rules:

1. If the file was lock-protected, the key name must agree with the lock name or start-up terminates. Start-up also terminates if the file-identifier field in the LBL job control statement does not match the VALUE OF ID statement specified in the DMCL source.

2. If you omit this statement, start-up only attempts to open those unlocked data base files in the DMCL.

3.3.1.2.9. QUICK-BEFORE-LOOKS Statement

The QUICK-BEFORE-LOOKS statement names the quick-before-looks file to be used for DMS run units accessing the data base defined by this DMCL. A separate quick-before-looks file must be assigned for each data base. The ON ERROR clause determines how I/O errors are processed if they occur while writing to the quick-before-looks file.

Format:

```
[QUICK-BEFORE-LOOKS FILE IS {lfd-name} [ACCESS-CONTROL KEY IS literal]
                            {DUMMY   }
      [ON ERROR {STOP   }].
                {PROCEED}
]
```

where:

lfd-name
: Is the same 1- to 7-character alphanumeric file name (the first character is alphabetic) that appears in the job's LFD job control statement and also in the DMCL source.

DUMMY
: Specifies that quick-before-looks processing is to be ignored. (In this case, you need not supply DVC and LFD job control statements for the quick-before-looks file.) The DUMMY option allows the automatic backward recovery facility to be turned off for this DMCL in a given DMS session.

literal
: Indicates the same name as the LOCK that was optionally applied to the quick-before-looks file in the DMCL source.

STOP
: Rolls back the run unit and shuts down the quick-before-looks partition. STOP is the default.

PROCEED
: Turns off automatic backward recovery, shuts down the partition, and allows the run unit to continue processing. Note that if a run unit requests automatic backward recovery after it is turned off, DMS issues a message to the operator's console indicating that manual recovery must be run against that data base.

Rules:

1. This statement is required only if the file is lock-protected or if you wish to override the ON ERROR option defined in the DMCL source.

2. The key name must agree with the lock name or start-up terminates. Start-up also terminates if the file-identifier field in the LBL job control statement does not match the VALUE OF ID statement specified in the DMCL source.

3. If the QUICK-BEFORE-LOOKS statement is omitted, the quick-before-looks file specified in the DMCL source is opened, if it is not lock-protected.

4. If the ON ERROR clause is omitted, the value assigned is the same value specified in the DMCL source.

NOTE:

You should use the DUMMY option very carefully so as not to compromise your data base. During DBMS start-up, automatic backward recovery normally examines the quick-before-looks file to see whether rollback is necessary because of a crash or some other contingency that occurred in the previous session. It will not make this check if you specify a dummy quick-before-looks file. No rollback will take place and, as a result, your users will be able to access a compromised data base.

For the DUMMY option to be used safely, both the previous and the current DBMS sessions must end normally. Otherwise, the data base may be in an inconsistent state, and you must recover it through some means other than automatic backward recovery. Moreover, any ROLLBACK or DEPART WITH ROLLBACK commands have no effect for this DMCL in a session for which you specify DUMMY.

The DUMMY option can serve a useful purpose by reducing the execution time of a lengthy program. For example, you might want to specify the DUMMY option when loading a data base immediately after initializing it with DBINT. If the program fails before it has completely loaded the data base, you must run it again from the beginning.

3.3.1.2.10. IMS QUICK-BEFORE-LOOKS Statement

The IMS QUICK-BEFORE-LOOKS statement names the quick-before-looks file used by IMS actions accessing the data base defined by this DMCL. The ON ERROR clause determines how I/O errors are processed if they occur while writing to the quick-before-looks file.

Format:

where:

`lfd-name`
Is the same 1- to 7-character alphanumeric file name (the first character is alphabetic) that appears in the job's LFD job control statement and also in the DMCL source.

`DUMMY`
Specifies that quick-before-looks processing is to be ignored for IMS transactions. Because IMS/DMS online recoveries are synchronized, you should be extremely careful in using this option. (See 3.3.1.2.9.)

`literal`
Indicates the same name as the LOCK that was optionally applied to the quick-before-looks file in the DMCL source.

`STOP`
Rolls back the transaction, shuts down the quick-before-looks file, and inhibits any IMPART statements for the DMCL. When file space is exhausted, DBMS rolls back the action, does not shut down the file, and does not inhibit IMPART statements. STOP is the default.

PROCEED
Turns off automatic backward recovery and continues processing the action. If, however, an action requests automatic backward recovery after it is turned off, DMS issues a message to the operator's console indicating that manual recovery must be run against that data base.

Rules:

1. This statement is required only if the file is lock-protected or if you wish to override the ON ERROR option defined in the DMCL source.

2. The key name must agree with the lock name or start-up terminates. Start-up also terminates if the file-identifier field in the LBL job control statement does not match the VALUE OF ID statement specified in the DMCL source.

3. If the IMS QUICK-BEFORE-LOOKS statement is omitted, the quick-before-looks file specified in the DMCL source is opened, if it is not lock-protected.

4. If the ON ERROR clause is omitted, the value assigned is the same value specified in the DMCL source.

3.3.1.2.11. CALC Statement

The CALC statement specifies the name of the CALC routine used by the run units accessing this DMCL.

Format:

```
[CALC ROUTINE IS {CALC00} .]
                 {CALC01}
```

Rules:

1. If this statement is omitted, CALC routine CALC01 is loaded.

2. The DBA should ensure that the CALC routine associated with a particular DMCL is not inadvertently changed at start-up time. If it is changed, the results are unpredictable.

3.3.1.2.12. ACTIVATE Statement

The ACTIVATE DMCL statement indicates the end of the optional statements for this particular DMCL and the beginning of the actual processing required to activate it.

Format:

```
ACTIVATE DMCL.
```

3.3.1.2.13. END Statement

The END statement is required to indicate the end of start-up syntax.

Format:

```
END.
```

3.3.1.3. Starting Up a User Data Base

This example illustrates a job control stream used to start up a user data base.

```
// JOB SPDBMS,,,21000
// DVC 20  // LFD PRNTX
// DVC 21  // LFD PRNTR
//DISK01 JSET  'DMS90X'
// DVCVOL &DISK01
// LBL 'DBA LIBRARY' // LFD LOAD
// DVCVOL &DISK01
// LBL 'SAMPLE DATABASE CUST'  // LFD DBMFIL1
// DVCVOL &DISK01
// LBL 'SAMPLE DATABASE PRODORD'  // LFD DBMFIL2
// DVCVOL &DISK01
// EXT ST,C,,CYL,4
// LBL 'SPDMCL QBL FILE'  // LFD SPQBLF2
// DVCVOL &DISK01
// EXT ST,C,,CYL,4
// LBL JOURNAL  // LFD SPJRNL
// EXEC DBMS,LOAD
/$
DBMS SECTION.
MAXIMUM DBMS RUN-UNITS IS 2.
JOURNAL FILE IS SPJRNL.
DMCL SECTION.
DMCL IS SPDMCL.
DATA BASE IS DBMFIL1 ACCESS-CONTROL KEY IS 'CUST'.
DATA BASE IS  DBMFIL2  ACCESS-CONTROL  KEY IS 'PORD'
ACTIVATE DMCL.
END.
/*
/&
```

3.3.2. DBMS Shutdown

Once loaded, the DBMS resides in main storage even though it may not be servicing requests at any given moment. In fact, it may have no active run units imparted or, may have no application programs loaded. When the DBMS reaches this quiescent state, and no further data base processing is planned, the DBA may terminate the current DMS session by keying in the following unsolicited message:

n0 SHUTDOWN DBMS.

where:

n

Is the job slot number of the DBMS; i.e., if DBMS executes under job slot 1, indicated on the system console, the command to be keyed in is:

10 SHUTDOWN DBMS.

The DBMS island code checks and when it confirms that no run units are active, it honors the shut-down request.

NOTE:

DMS treats IMS as a special case. If IMS action programs have issued calls to DMS, then IMS is "logged on" to DMS. DMS will honor your shutdown request even with IMS still in the system, provided no action programs have currently active calls to DMS. This allows you to shut down DMS in an orderly fashion while still letting IMS continue with its non-DMS work.

3.4. DEPART CHECKPOINT STATISTICS

DBMS loads the subschema and DMCL load modules at IMPART time, giving the application program a tie to the logical and physical descriptions of the data base.

After COBOL/DML application program execution, DEPART checkpoint statistics are logged for each program (run unit). Figure 3-2 is a sample DEPART checkpoint statistics listing. Table 3-1 describes the meanings of the checkpoint statistics.

```
   DMS/90 STATISTICS FOR SPROGM
NQBL.....18
NBPG.....18
NAPG.....60
PGIN.....171
PGOUT.....60
PGUPD.....18
CPGS.....21
CPGD.....0
VPGS.....30
VPGD.....0
RECRQ.....609
CURRU.....255
NCALL.....269
```

Figure 3—2. Sample of DEPART Checkpoint Statistics Listing

Table 3—1. DEPART Checkpoint Statistics

Name	Description
NQBL	The number of before-page images written to the QBL file.
NBPG	The number of before pages written to the journal file.
NAPG	The number of after pages written to the journal file.
PGIN	The number of times a page was read from the data base into the DBMS common buffer pool. Note, a requested page may already be in a buffer read by a concurrent task.
PGOUT	The number of times a page was written to the data base from the DBMS common buffer pool. Note, a task may write a page that another task updated.
PGUPD	The number of different data base pages updated.
CPGS	The number of times that a record with location mode CALC was stored on the page that the CALC algorithm produced.
CPGD	The number of times a record with location mode CALC could not be stored on the page that the CALC algorithm produced.
VPGS	The number of times that a record with location mode VIA or DIRECT was stored on the same page as its set owner record, or as defined by DIRECT-DBK.
VPGD	The number of times that a record with location mode VIA and DIRECT was stored on the same page as its set owner record, or as defined by DIRECT-DBK.
RECRQ	The number of internal record requests.
CURRU	The number of records that became current of run unit during the execution of the program.
NCALL	The number of calls to the data base management system (DBMS). That is, the number of DML verbs that were actually executed during the running of the application program.

3.5. DBMS BUFFER MANAGEMENT

The DBMS maintains a pool of data base buffers and space inventory buffers for each data base (DMCL). The buffer pools are shared by all run units that sign onto a data base. The buffers are effectively managed on a history access queue according to most-recently-used and least-recently-used status. When a buffer is required for input, the least recently used buffer is selected.

Record updates are made to buffers without immediately writing them to disk. An updated data base page (or space inventory page) is written to disk only when the buffer is required for a new data base page or when an updating run unit is in the process of departing or closing an area. When a data base page is written to disk as a result of departing, the image is still maintained in the buffer for possible reference by other run units.

4.3. AUTOMATIC BACKWARD RECOVERY

Automatic backward recovery (ABR) is an online recovery process that restores the data base by rolling back the uncommitted updates. Uncommitted updates are those made since the most recent checkpoint or since the beginning of the run unit. Automatic backward recovery is classified into two distinct processes:

1. Run-unit rollback
2. Quick recovery

The run-unit rollback process applies before-looks contained in the quick-before-looks partition to the data base. DBMS takes before-looks of data base pages before any data base page is updated and writes the before-looks to the quick-before-looks file (a SAT partitioned file).

DBMS performs the run-unit rollback for any run unit for a variety of reasons (e.g., deadlock, DBMS internal error, or run-unit error). The run-unit rollback may be initiated by any of the following:

- DEPART WITH ROLLBACK command
- ROLLBACK command
- Forced DEPART by DBMS

After successful rollback, the quick-before-looks partition assigned to the run unit is marked empty by writing a rollback checkpoint and is made available to other run units, except when the ROLLBACK command is used.

On the other hand, quick recovery is initiated at DBMS start-up time to reset the data base to the last consistent state. Quick recovery may be considered a warm restart process that triggers multiple run-unit rollbacks.

Quick recovery opens the quick-before-looks file specified via DBMS start-up JCL and examines whether there are any active quick-before-looks partitions that must be rolled back. For each active partition, quick recovery performs a run-unit rollback and marks the partition logically empty by writing a rollback checkpoint after the successful rollback. If any error occurred during this process, an error message is displayed and no attempt is made to roll back remaining active partitions for the same data base.

After successful quick recovery for the data base, a message is displayed and the processed quick-before-looks file is reinitialized and reopened with new partitions that match the number of updating run units for the current DBMS session. The quick-before-looks file stays open until the DBMS job is shut down. If more than one DMCL is specified in the current DBMS session, the quick-before-looks file specified for each DMCL is examined to see if quick recovery is required for the data base. If any quick recovery process failed, DBMS terminates at this point. (Refer to the IMS/DMS interface user guide/programmer reference, UP-8748 (current version), for information about automatic backward recovery processing of the IMS quick-before-looks file.

4.3.1. Quick-Before-Looks File Allocation

If you use automatic backward recovery, the DBA must name in the DMCL and allocate a unique quick-before-looks file for each data base. The space allocated to the quick-before-looks file must be large enough to contain the before looks for all updating run units executing against a given data base. The quick-before-looks file should also be allocated with file extension in case the primary space allocation is insufficient.

The DMCL for a DBMS job identifies a quick-before-looks file via an lfd-name and an lbl-name or file-identifier. DMS validates the quick-before-looks filename during DBMS start-up and quick-before-looks file OPEN processing. If the filename in the DMCL does not match the filename of the quick-before-looks file specified to DBMS in the job control stream, a fatal error results and start-up terminates.

Avoid scratching or reinitializing the quick-before-looks file or files (using // SCR or // LFD filename,,INIT) unless you are reloading the data base after a system crash. Scratching or reinitializing the quick-before-looks file causes the quick recovery process to be bypassed and as a result may leave the data base in an inconsistent state. The DMS manual recovery utilities, DBRES and DBREC, logically scratch the quick-before-looks files following their successful recovery process.

No quick-before-looks file allocation and no quick-before-looks file specifications in the DMCL are needed if quick before looks are not specified for any area in the DMCL.

Refer to the IMS/DMS interface user guide, UP-8748 (current version), for information about allocation of the IMS quick-before-looks file.

4.3.2. Quick-Before-Looks File Management by Automatic Backward Recovery

DBMS partitions the quick-before-looks file by assigning each updating run unit a quick-before-looks partition. A maximum of seven partitions can be assigned. This sets a limit of seven updating run units which can concurrently execute against a given data base.

If a fatal I/O error occurs during the quick-before-looks file write operation, DMS displays a message describing the error condition and shuts down the quick-before-looks partition.

You are given two options to recover from this error via either the DBMS start-up syntax or the DMCL specification. The ON ERROR STOP/PROCEED option of the QUICK-BEFORE-LOOKS statement in the DBMS start-up syntax overrides that of the QUICK-BEFORE-LOOKS statement in the DMCL specification.

- If the ON ERROR PROCEED option is in effect for the current DBMS session, the run unit is allowed to continue executing. In this case, DMS displays a console message to indicate that automatic backward recovery was disabled for the run unit. If this run unit successfully completes a DEPART or FREE WITH CHECKPOINT process afterwards, DMS displays another console message indicating that automatic backward recovery was enabled again for the run unit. If a run unit attempts a rollback when automatic backward recovery is disabled for the run unit, DMS displays console messages stating that the data base must be recovered offline by executing DBREC.

- If the ON ERROR STOP option is in effect for the current DBMS session, the DMS cancels the run unit. This causes an immediate rollback.

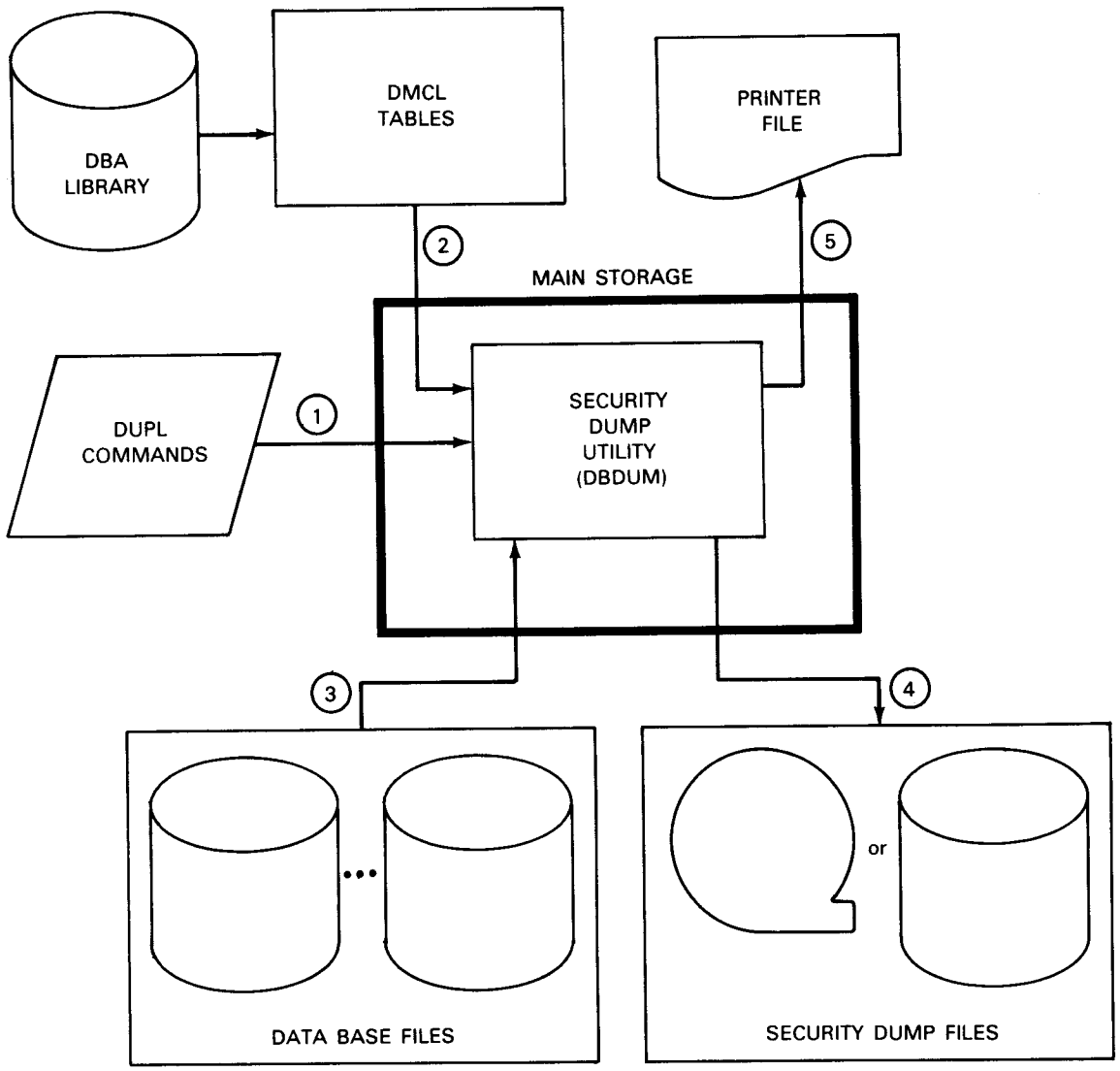


Figure 4—4. Data Base Security Dump Utility Operations (DBDUM)

4.5.1. Security Dump File Format

The security dump file is a SAT file directed to tape or disk by the user-issued // DVC // LFD job control sequence. The first record in the security dump file is a header record that identifies the total contents of the file. The header record contains the DMCL name, creation date/time stamp, and type of requests. (See Figures A-8 and A-9 for security dump file record formats.)

To identify your dump request, a separator record prefixes each explicitly dumped portion of the data base. The separator record is a modified header record the same size as the data base page.

Because you may request dumps for all implicit data bases as well as parts of a data base (explicit) including areas and pages, portions of the data base may be duplicated on the same dump file.

DBDUM executes the DUMP command by writing each portion of a data base to the dump file in the same order the user specified on his DUMP command. Because DBDUM operates in this way, you can format a dump file to your specific needs. For example, place small or more commonly restored segments at the beginning of the dump file. If there are any processing errors during the execution of DBDUM or if the job is cancelled, the header record and the last separator record will be inconsistent with the actual contents of the dump file.

4.5.2. DUMP Command

The DUMP command requests a dump of a page range, area, or entire data base to a specific file, and optionally specifies the creation of a data base utilization report. (See 4.5.3.)

A DUMP command may contain one or more DUMP clauses, each containing one or more requests. All requests within each DUMP command define the contents of one security dump file. You may issue any number of DUMP commands in one execution of DBDUM, thus creating a separate dump file for each complete DUMP command.

DBDUM begins actual processing when it detects a period at the end of the command. Thus, when DBDUM validates more than one DUMP clause, each ending with a semicolon, DBDUM stacks the information and processes the dump when it detects the period at the end of the command.

During DUMP statement validation, if a page range or area is invalid or inaccessible, DBDUM sends an error message to the printer file and ignores that DUMP statement. If there are multiple requests within a DUMP statement (DUMP AREAS ONE, TWO, THREE ...) and one is invalid, DBDUM validates the remaining requests but ignores the entire statement during processing.

4. Restore the data base.

5. Logically scratch the quick-before-looks files.

6. Generate the printer output file containing DUPL input commands, the messages monitoring execution of DBRES, and (optionally) a header record display.

Figure 4-7 is an example of the printed report generated by DBRES.

4.6.1. RESTORE Command

Format 1 of the RESTORE command specifies the name of the security dump file from which DBRES restores all or portions of a data base. Format 2 specifies what is being recovered; i.e., the entire data base, areas, or pages. Deferred processing is available for format 2 of the RESTORE command. In this case, DBRES validates RESTORE clauses terminated by a semicolon and stacks them until it receives the final RESTORE clause that ends with a period.

If there are multiple restore requests for the same area name or page range within one complete command, DBRES accepts the requests but restores the area or page only once.

In addition to updating the space inventory record for each page that is restored, DBRES places the name "DBRES" in the run-unit ID of the block header of each data base page restored.

Format 1 Operation:

The RESTORE FROM command opens the security dump file specified by the lfd-name, validates it as the correct security dump file, and (if so) reads the header records into the job region work area.

The ON ERROR clause determines whether DBRES accepts the next statement or terminates processing. DBRES also references the ON ERROR clause during the actual processing phase. In general, if errors occur during open or close, the ON ERROR option setting determines whether the utility terminates or continues; however, on I/O errors, DBRES terminates abnormally with a dump.

Format 2 Operation:

Format 2 ensures that the requested content-description is contained within the dump file. If validation errors occur, DBRES validates any remaining portions of the command but does not process the entire command.

Format 1:

```
RESTORE FROM lfd-name[USING DMCL dmcl-name][{ON ERROR STOP   }]-
                                            {        PROCEED}
```

```
CU01  START OF DMS90 UTILITY * DBRES * VERSION 07.1


DMCL IS SPDMCL.
CU03  DMCL TABLES SUCCESSFULLY LOADED
DATA BASE FILE IS DBMFIL1 ACCESS-CONTROL KEY IS CUST.
DATA BASE FILE IS DBMFIL2 ACCESS-CONTROL KEY IS PORD.
ACTIVATE DMCL.
DISPLAY FROM DUMP02.

SECURITY DUMP FILE GENERATED:   81-03-06/12:11:53.000

DMCL NAME IS:   SPDMCL00

.....SEGMENT DUMPED.....      DUMPED BY CONTROL STATEMENT
        1 THRU 20                    DUMP PAGE(S)
CUSTOMER-AREA                      DUMP ALL AREAS
ORDER-AREA                         DUMP ALL AREAS
PRODUCT-AREA                       DUMP ALL AREAS

CR14  END OF HEADER DISPLAY

RESTORE FROM DUMP02.
RESTORE ALL AREAS.

DATABASE RESTORED AT: 81-03-26/03:48:26.000,  LBL IS: DUMP2
RESTORED 10 PAGES, FROM 1 THRU 10 INCLUSIVE ( AREA IS CUSTOMER-AREA    )
RESTORED 10 PAGES, FROM 11 THRU 20 INCLUSIVE ( AREA IS ORDER-AREA      )
RESTORED 10 PAGES, FROM 36 THRU 45 INCLUSIVE ( AREA IS PRODUCT-AREA    )

END.
CU02  END OF DMS90 UTILITY * DBRES * VERSION 07.1
```

Figure 4—7. Example of DBRES Report

where:

lfd-name
: Is the user-supplied file name of the security dump file as it appears in the // LFD statement of the job control stream.

dmcl-name
: Is the name used to create the dump file as it appears in the header record of the dump file.

The USING clause lets you restore a data base using a DUMP file created from a different data base (DMCL). When this clause is specified, DBRES verifies the dmcl-name with that contained in the dump file header record. During the restoration process, DBRES changes the dmcl-name in each data base page it restores to the name specified in the 'DMCL IS' command.

This allows you to dump a production data base and restore it to a test data base with a minimum of effort. It is your responsibility to ensure the data bases (DMCLs) are the same except for the dmcl-name.

Format 2:

```
RESTORE content-description[;RESTORE content-description].
```

where content-description is:

```
{ {PAGE  }page-number-1[THRU page-number-2]  }
{ {PAGES }                                   }
{ALL AREAS                                   }
{ {AREA  }area-name-1[,area-name-2]...       }
{ {AREAS }                                   }
```

Rules:

1. You must supply formats 1 and 2 of the RESTORE command to restore any data base or portion of a data base.

2. The value of page-number-1 must be less than or equal to page-number-2.

3. If the THRU clause is omitted, only one page is restored.

4. The page range specified must be completely contained in an explicitly dumped range.

5. If any area is shut down, DBRES does not attempt to restore the data base.

6. ALL AREAS assumes the dump file contains a DUMP ALL AREAS.

7. The maximum number of area names that can be processed by a single AREA statement is 10.

The job control stream needed to execute the data base security restore utility (DBRES) follows. AREA3 is stored from the DUMP security file.

integer-1
Is a decimal number and a multiple of 4 (or zero). It represents the number of bytes offset from the beginning of the page to apply the patch.

literal-1 and literal-2
Are hexadecimal values which are eight characters long. They represent the current contents and the requested patch.

Rule:

1. ON ERROR clause operates as described in 1.4.

Job Stream Example:

The following example illustrates a request for a change to data base page 1216. If the 30th full word (at 116 bytes offset) of that page is X'001A076F', that word on data base page 1216 will be altered to X'001A376F'.

```
// JOB ALTER
// DVC 20 // LFD PRNTR
// DVCVOL &DISK01
// LBL'DBA LIBRARY'// LFD DBAL
// DVCVOL &DISK01
// LBL'SAMPLE DATABASE CUST'// LFD DBMFIL1
// DVCVOL &DISK01
// LBL'SAMPLE DATABASE PRODORD'// LFD DBMFIL2
// EXEC DBPAG,DBAL
/$
   DMCL IS SPDMCL.
   DATA BASE FILE IS DBMFIL1 ACCESS-CONTROL KEY IS CUST.
   DATA BASE FILE IS DBMFIL2 ACCESS-CONTROL KEY IS PROD.
   ACTIVATE DMCL.
   ALTER PAGE 1216 STARTING AT 116 BYTES OFFSET
   REPLACING 001A076F WITH 001A376F.
   END.
/*
/&
// FIN
```

5.1.5. NFP FILE Command

The NFP FILE command identifies the file that contains the numeric field parameters (NFP) to be applied for printing data base pages.

Format:

```
NFP FILE IS lfd-name MODULE module-name [ON ERROR {STOP    }] .
                                                  {PROCEED}
```

where:

lfd-name
: Is the name on the job's LFD statement.

module-name
: Is the name of an OS/3 librarian source file containing the numeric field parameters (5.3).

Rules:

1. Only one NFP command is allowed per data base page dump run.
2. If used, this command must come after the DMCL command and before the ACTIVATE command.

5.2. JOURNAL FILE AUDIT UTILITY (JFAUD)

JFAUD is an independent utility that generates the following reports based upon journal file information:

- Updates of a page
- Status of a day's activities
- Journal file records
- Journal file checkpoints

These reports aid the DBA in maintaining data base integrity. Because journal files contain a history of all activities against the data base, by running the JFAUD utility, the DBA can make appropriate decisions concerning data base maintenance and recover. Figure 5-3 identifies the functions performed when the JFAUD utility is executed (see circled numbers):

1. Initiate JFAUD via DUPL commands.
2. Load the DMCL tables.

Appendix A. Data Base and Journal Formats

A.1. DATA BASE FILES

A DMS data base consists of one or more OS/3 system access technique (SAT) files. The data base records are grouped into numbered pages, which are related to relative block numbers in the file. The first page number is 1.

Data base files include two kinds of pages - data base pages and space inventory pages. The number of data base pages is allocated in the device media description. (See the data description language user guide/programmer reference, UP-8022 (current version).) The number of pages specified does not include the required space inventory pages. The size of a data base page is also specified in the device media description and must be a multiple of 2048 bytes.

A.1.1. Allocating Data Base Files

The SAT files on which a data base resides must be preallocated before executing DBINT or DBRES to initialize or restore the entire data base. The file allocation must include sufficient space to contain both the data base pages and the space inventory pages.

To compute the number of space inventory pages needed in a data base, use the following formula. Use integer arithmetic, discarding the fractional portion of the quotient.

$$\left[\frac{2n - 1}{p} + 1\right] = \text{number of space inventory pages}$$

where:

n
: Is the number of data base pages declared in the device media description.

p
: Is the size of a data base page declared in the device media description.

For example, if you have declared 2000 pages of 4096 bytes each, you need:

$$\left[\frac{2 \times 2000 - 1}{4096} + 1\right] = 1$$

The high page number for each data base file is computed and displayed by the DMCL processor during the DMCL compilation. This figure may be used to allocate file space.

A.1.2. Data Base Page Format

Associated with each data base record is a unique number called the data base key. The data base key is used to locate the record and is, in effect, a pointer to the record. This key remains unchanged throughout the life of the record. When a record is deleted, its data base key is then available for use by a new record.

The data base key is a binary number stored as a full word (four bytes). The most significant 23 bits following the sign represent the page number. The least significant eight bits represent the record number or line number within the page. This representation allows for up to 8,388,607 pages and up to 255 records in each page. The actual number of records stored in a page is, of course, dependent on the size of the records. The first 100 bytes contain page-related and journaling data.

Each data base page is initialized to a specific format by the data base initializer utility program (DBINT). The first 16 bytes of the page following the first 100 bytes of journal data are the page header. The last eight bytes of the page are the page footer, which contains page and line information. A line entry array consisting of 8-byte elements is built backwards from the page footer. The line entry array contains one element for each record stored in the page. Figure A-1 shows the data base page layout.

The page header consists of the page number followed by a system data base which is referenced as line 0 and is the owner of the CALC set for the page. Thus, all records that CALC to a given page are chained together to form a set that is owned by the page header record for that page. This allows a CALC record to be found regardless of its physical location. The need for specifically designated overflow areas is eliminated and the value of the CALC field in a record may be changed without changing the physical location (and the data base key) of the record.

The page header is defined as follows:

Displacement	Length	Description
0	4	Page number
4	4	Next pointer for CALC set (start of system record, line 0)
8	4	Prior pointer for CALC set
	2	Available space in page
14	1	Journal file must write switch
15	1	Data base must write switch
	16	

The page footer is defined as follows:

Displacement	Length	Description
0	2	Line space used
2	2	Reserved
4	4	Page number
	8	

The line entry array element is defined as follows:

Displacement	Length	Description
0	2	Record ID and "deferred-delete" flag
2	2	Displacement to record within page
4	2	Total record length
6	2	Displacement to data within record
	8	

Note that the page number appears in the first and last full word within the page. This is for validity checking.

Each record stored in a page has a corresponding line entry array element. Records are located within the page by using the line number portion of the data base key as an index into the line entry array. A line entry array element does not exist unless the corresponding record exists in the page.

When the leftmost bit of the record id field is set to 1, the record is considered logically deleted. The data portion of the record has been released, but linkages remain. The final deletion of the record space (linkages) is deferred until the record can be efficiently delinked from all sets in which it is a member.

When you delete a record, all records in the affected page are compacted toward the top of the page. Thus, all available space is a contiguous block at the bottom of the page. Because records are located through the use of the line entry array, there need be no specific order of storage of the records. All new records are stored into the available space, following the last record, after any necessary compaction has been done.

A record cannot span pages, nor can a record's length be changed.

The line 0 element in the line entry array (8 bytes) is always required with the journal data (100 bytes), the page header (16 bytes), and page footer (8 bytes). The total page overhead is 132 bytes.

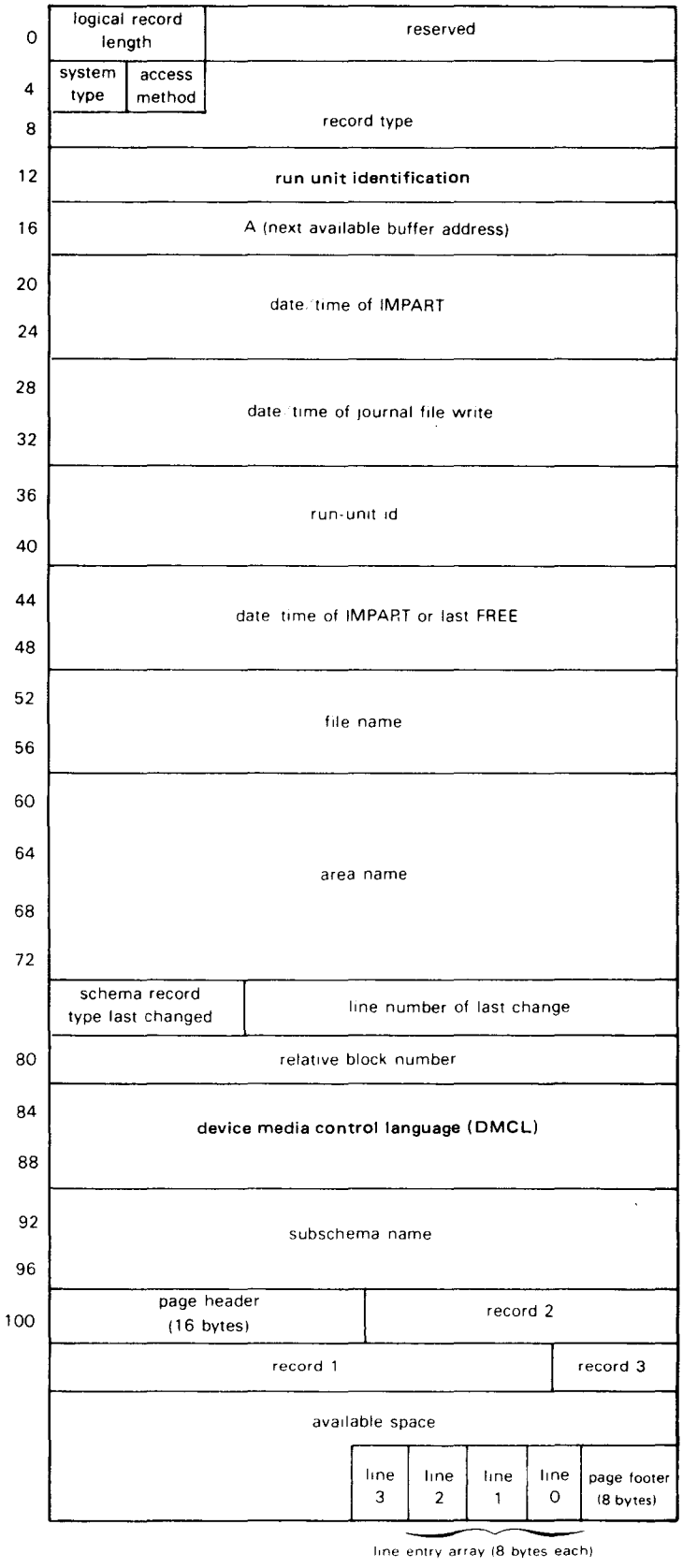


Figure A—1. Data Base Page Layout

A.1.3. Space Inventory Pages

Data base files contain space inventory pages, as well as data base pages. The space inventory page (same size as the data base page) contains information used by the DMS space management algorithm. The entire page contains half-word entries, each of which specifies the available space in its related data base page. These half words are called space inventory entries. Figure A-2 shows the space inventory page (SIP) format.

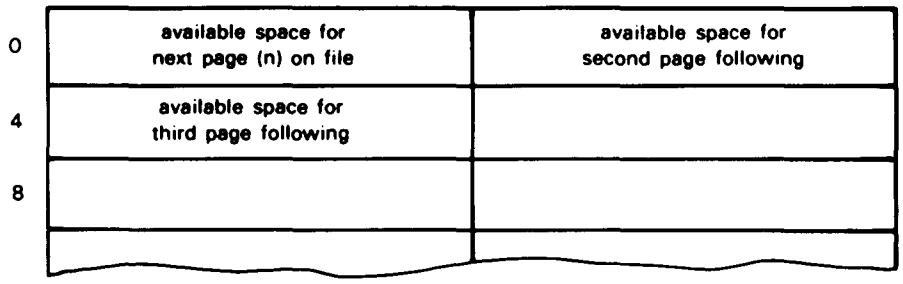


Figure A—2. Space Inventory Page Format

Each space inventory page contains space inventory entries for a number of data base pages. The number of pages represented on each space inventory page is:

$$\frac{\text{page-length}}{2}$$

The space inventory page precedes the data base pages for which it contains information. If the page length is 2048, the first space inventory page represents the following 1024 data base pages. The second space inventory page represents the next 1024 data base pages. Each data base file begins with a space inventory page that relates to the data base following it. Since data base areas may span files, the page numbers on a file need not be continuous. Therefore, accesses to the file are based on a relative block number, starting with 1. Each block is the size of a data base page. Figure A-3 shows the SIP placement for files on which the data base page size is 2048 and 6144 bytes.

DMS calculates the proper space inventory page based on the relative block number of the data base page to be used for storage. (DMCL tables are used for this information.) The relative block number is then used to find the relative offset within the space inventory page to obtain the half word related to the requested page.

To minimize the number of I/O accesses, the space inventory page is not updated every time a data base page is updated. The space inventory page is updated when:

- a STORE operation is performed and there is no room on the data base page to store the record; or
- a DELETE operation is performed and the data base page is at least 50 percent full.

When a STORE operation is performed and there is insufficient room on the data base page, the space inventory entry for that data base page is updated and the space inventory entries for succeeding data base pages are accessed until a page with available space is found. If DBMS reaches the end of the area without finding a page with available space, the search resumes at the beginning of the area and continues until the starting point is reached.

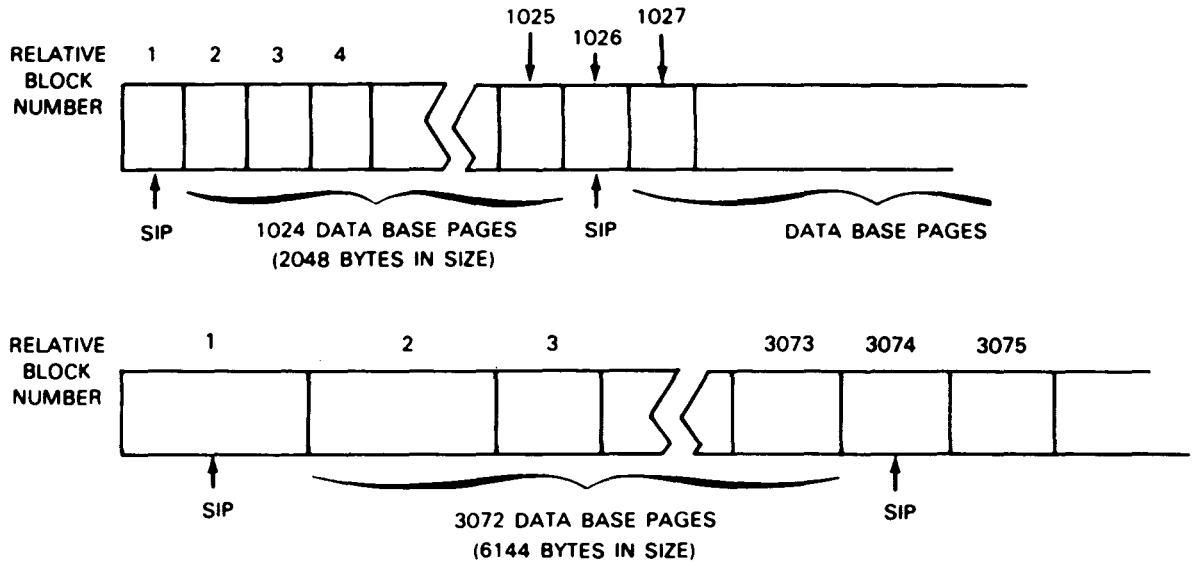


Figure A–3 Space Inventory Page Placement on a Data Base File

Several data base utility programs affect the space inventory pages. These are DBINT, DBRES, and DBREC. DBINT resets the entries in the space inventory pages to the maximum data space. DBRES and DBREC store (into the space inventory page) the actual available space in the page that is being restored to the data base.

Note that a space inventory entry is not meant to be a completely accurate indicator of space available, but only a guide. Also, if the space inventory entries become invalid for any reason, the system will continue to function and ultimately correct any inaccuracies without the necessity of outside intervention.

A.2. DMS JOURNAL FILES

DMS utilities use the journal file to recover from hardware or software errors that might occur while a run unit was updating the data base. DBREC reads the journal tape backwards and writes the "before" images to the data base for all the updated pages. The backward processing ensures that the earliest before images are restored to the data base.

DBREC also reads the journal tape forward and writes the "after" page images to the data base for all the updated pages or for only those pages selected by the user. The forward processing results in the latest after-page images being stored in the data base.

The DMS journal tape consists of two types of records:

1. Checkpoint records that identify specific actions made on the data base

2. Data base page images

Figures A–4 through A–9 provide detailed information on each type of journal file record.

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD