KMSYS
WORLDWIDE, INC

Database Reorganization User Guide

10 August 2011

The information contained in this document is the latest available at the time of preparation; therefore, it may be changed without notice, and it does not represent a commitment on the part of KMSYS Worldwide, Inc. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than stated in the terms of the agreement, or without the express written permission of KMSYS Worldwide, Inc.

If you have any comments about the software or documentation, notify KMSYS Worldwide, Inc., in writing at the following address:

KMSYS Worldwide, Inc.
P.O. Box 669695
Marietta, Georgia 30066
U.S.A.

Technical Support (770) 635-6363 - Main Number (770) 635-6350 - Fax (770) 635-6351

I-QU PLUS-1 Release 11R6, November 1999

Table of Contents

Chapter 1: Introduction

The Database Reorganization User Guide will discuss the database reorganization features of I-QU PLUS-1. This Guide demonstrates how I-QU PLUS-1 may be used to perform large, complex database reorganizations quickly and efficiently. While this document is primarily directed to the database administrator or database analyst, it will demonstrate that use of this product and the associated programming language brings complex database maintenance and reorganizations within the reach of the less than "expert" DMS 2200 internalist.

I-QU PLUS-1 was designed as a high level, multi-mode database query and update processor. The processor is multi-mode in that it provides the COBOL/DML programmer with an interactive DML access method; it provides the user support organization with a method for retrieving database information for ad hoc reporting; and it provides the database specialist a high level language for performing database reorganizations. Isolating target areas or structures, while maintaining logical relationships outside the target area or structure, is the key feature of the I-QU PLUS-1 reorganization method.

I-QU PLUS-1 can access DMS 2200 hierarchical databases RDMS 2200 relational databases, PCIOS files, MAPPER reports via DTM, DB4 databases and TIP files. No special subschema or pre-definition of files is necessary. Any DMS 2200, COBOL subschema may be dynamically invoked, and once invoked, the interactive I-QU PLUS-1 user or batch program has access to all database data names and data item names defined in the subschema. The user may then interface with DMS 2200 as any standard run unit. All activity is audited normally (in the multi-thread DMS 2200 environment), quick-look and rollback protection are operational, and any specified data base procedures will be invoked by the DMR.

When I-QU PLUS-1 commands are entered interactively, error status information is displayed immediately. The interactive capabilities of the product make it possible to browse and/or update all supported file systems. This capability also allows COBOL/DML programmers to replay program logic interactively, command by command. In batch mode, the complete programming language provides a SORT verb, data item definition, integer arithmetic functions, logic functions, structured programming constructs, and procedure definition.

Chapter 2: Purpose of a Reorganization Utility

The need for database reorganization has existed since databases were first developed, even before data management systems. As data is stored in some defined relationship to other data, links are established to facilitate the retrieval of elements throughout the structure. The database management system (DBMS) must employ some algorithm to search for space when new elements are to be added to existing structures, and when new occurrences of the structure are added. The deletion of individual elements within a structure requires the DBMS to perform all necessary operations to ensure that the logical relationship of remaining members is maintained. The deletion of elements within a structure, or the deletion of entire occurrences of a structure, results in the fragmentation of the storage media, which may or may not be efficiently reused by the DBMS storage algorithm.

Fragmentation of the storage media will result in both decreased database performance and an increase in the amount of storage required for a fixed amount of data. Over time, database areas and structures will tend to degenerate due to the addition, deletion and modification of data. Reorganization is the method employed to optimize database performance and mass storage utilization. Reorganization will also be required if certain types of physical or logical changes are applied to an existing database. The purpose of reorganization in this situation is to establish required links and/or physical attributes.

2.1 Conventional Tools

According to specifications, the Database Reorganization Utility (DRU), as supplied by Unisys, will perform some physical and logical reorganization tasks. For example, the size of an area may be changed, an area mode may be changed from SARP to DARP, or area codes may be changed. DRU, however, has some limiting factors. When a new set relationship is to be added to an existing database, DRU cannot be used. Nor can DRU handle changes in the length and/or form of data within records. Indeed, DRU cannot handle many forms of such "logical" changes to the database. DRU was designed for the user with a very limited knowledge of the internal structures used by DMS 2200, and is therefore not very flexible.

Conventional COBOL/DML programs can be used for reorganization, doing an unload under the old schema and reloading under the new schema. While this will accomplish what is necessary, it is costly in terms of program development time. Also, down time due to reorganization will become a function of database design, in that, the primary target of the reorganization (i.e., the records and sets most needing reorganization) will very likely be logically joined to other areas and records. Even though structures, which are logically joined to the main object of the reorganization, may not need reorganization, they too will have to be unloaded and reloaded to preserve defined relationships. This constraint will not only lengthen the time required for reorganization, but will also increase the complexity of the program and the probability of error.

Consequently, the network database architecture supported by DMS 2200 can be viewed as a double-edged sword. On the one hand, it is a powerful method of structuring data based on its use and relationship to other data; on the other hand, optimum design decisions regarding performance and ease-of-use are necessarily compromised in order to arrive at a design that can be realistically maintained. Even with careful planning, the "real world" often wins out and forces unanticipated relationships to be tacked on to our (otherwise perfect) initial design.

2.2 I-QU PLUS-1 Advantages

I-QU PLUS-1 was designed with the above-mentioned considerations in mind. This reorganization tool will:

- Allow reorganization of any subset of the database while maintaining logical relationships outside the target structure;
- Allow new pointers to be added into existing relationships (i.e., owner, last and/or prior pointers);
- Allow new relationships (sets) to be added without forcing the reorganization of all areas and records in the structure;
- Allow the expansion and contraction of data fields and records without reorganization of unaffected record types.

The capability provided through I-QU PLUS-1 will allow complex networked data structures to be designed without fear of complex reorganizations. Reorganizations may be divided into small, manageable segments thus reducing down time and increasing the likelihood of success. Database design can also be optimized based on present requirements. Because the introduction of new record types and/or relationships is facilitated, current ease of use or performance need not be compromised in anticipation of "future needs". And, because reorganizations can be done in small segments, database performance and mass storage utilization can be maintained nearer a peak level.

Chapter 3: I-QU PLUS-1 and the Utilities

3.1 Method of Reorganization

I-QU PLUS-1 is able to accomplish the isolation of targeted data structures through the use of commands which manipulate database pointers. These commands allow database administration personnel to retrieve specified logical links (pointers) from the DMS page buffer. Once retrieved, the pointers may be sorted, stored, cross-referenced or otherwise processed. Once all processing is complete, the pointer may be reinserted into the proper location on the DMS2200 page in the page buffer. This capability is used by I-QU PLUS-1 to pull logical links (database pointers) for those relationships (sets) not targeted for reorganization out of the object records at unload time, and restore these links at reload time. All this is accomplished using the high-level programming language of I-QU PLUS-1 and a set of powerful supporting programs that make up I-QU PLUS-1.

3.2 Supporting Utility Programs

Since I-QU PLUS-1 is designed to isolate areas being reorganized from other affected areas (areas containing records that point into the reorganized area, or records that are pointed to by the reorganized area), two programs have been provided to manipulate record pointers within the database without use of the DMR. The first of these programs, Pointer Build (PBLD), is used to build a cross-reference index of old database pointers to new database pointers. The cross-reference index is then used by the second program, Pointer Fix (PFIX), to RELINK affected areas back to the reorganized areas after the reload operation. PFIX actually does the RELINK of set relationships broken in order to speed up reorganization. This program uses MASM subroutines to read database pages directly, make pointer changes using the pointer cross-reference file, and rewrite the page, independently of the DMR. PFIX also has the capability to perform two other functions: DELINK and DBPCHANGE. The DELINK function is used on owner records to make set occurrences empty when the members reside in a different area and the members' area is to be initialized and reloaded. DBPCHANGE allows area codes and database pointer formats to be changed without performing an actual reorganization.

In addition to the PBLD and PFIX utilities, two other programs are provided to simplify the setup and execution of reorganization:

1. Query Schema (QRYSCH) -a program used to query various object schema tables to determine the absolute position of set pointers and other attributes important to the planning of a database reorganization. This program is also used to obtain information required in specifying pointer manipulation commands in the I-QU PLUS-1 programs.
2. Schema Utility (SCHUTL) -a program used to directly modify the schema absolute. The use of SCHUTL saves time otherwise spent compiling special load schemas

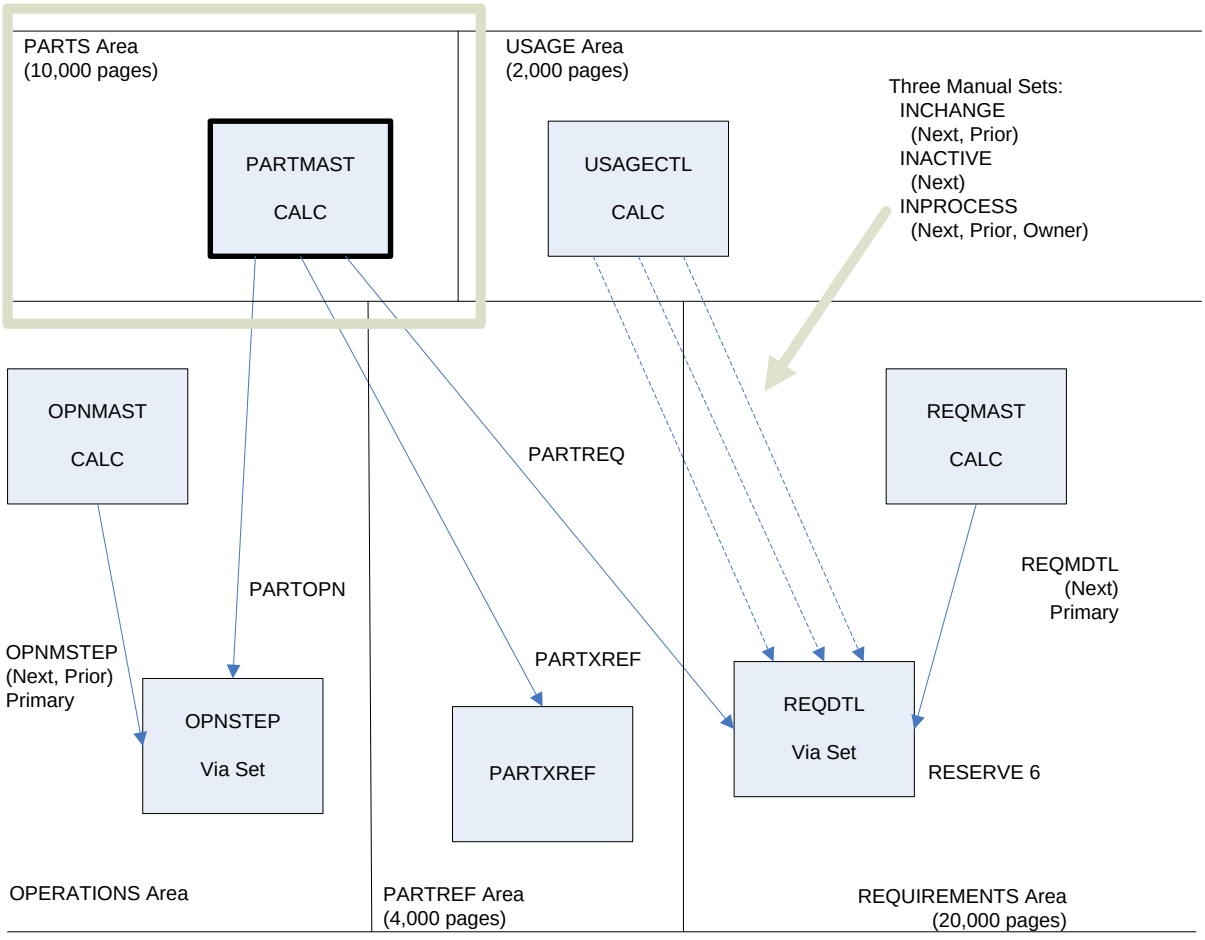
necessary to gain optimum speed for database reloads. SCHUTL supports several functions:

- Change area looks to NO-LOOKS in order to gain optimum speed during a database reload operation.
- Change PREINITIALIZED pages to non-preinitialized so that a DMU initialize of an area may be skipped during the reorganization process.
- Force a record's participation as a member of a particular set to be ignored by the DMR when the record is stored. This method can be used to gain a great deal of speed under certain circumstances.
- Change the TIP schema file code or EXEC file name in an object schema and subschema in order to direct the DMR to retrieve the modified object schema and subschema from an alternate TIP file upon IMPART. This function allows I-QU PLUS-1 programs to run independently of the normal production environment using the schema changed by this utility.
- Change all TIP areas to EXEC areas to allow complete independence from the production environment when testing or actually performing database reorganizations. This command also changes TIP schema files to EXEC and can be used in conjunction with the preceeding function.
- Change a CALC record's definition to ignore the check for duplicate records when reloading the CALC area. This command inhibits the DMR code from searching an entire CALC chain each time a CALC record is stored, thus leading to possible savings in I/O – especially if CALC records tend to overflow the prime page.
- Change the LOAD factor to allow room on pages to store records during update. By changing the load factor in this manner, the overhead otherwise required to compile a special LOAD schema is eliminated.
- Change set order to NEXT to avoid the overhead (I/Os) associated with set ordering during reload. This method can be beneficial when a primary set in the production environment is order FIRST or SORTED. Changing the set to order NEXT can reduce the number of I/Os required to link members into a set occurrence when the set occurrence spans multiple pages.
- Ignore RESULT processing on owners when reloading members, thus eliminating I/Os to the owners.
- Change set occurrence selection from LMO to COS, thus eliminating expensive navigational I/Os otherwise required to store members.

The use of these programs will be demonstrated by example several times in the remainder of this manual. The application of each program will be explained in the reorganization examples.

Chapter 4: Database Reorganization Example One

To describe how I-QU PLUS-1 is used, a realistic reorganization will be discussed. This example is not the most complex of situations, but it will demonstrate the overall use of I-QU PLUS-1. Consider the following portion of a manufacturing database:



4.1 General Reorganization Process

The procedure followed for most database reorganizations is quite simple: the area is unloaded under the old schema, the new schema is swapped in, and the area is reloaded. Since I-QU PLUS-1 does not require the unloading of all logically linked areas, one additional step must be performed. Links in affected areas that point to records in the reorganized area, will have to be resolved. These pointers must be changed to point to the new location of the reorganized records. In addition, pointers in the reorganized area that point to other areas will be replaced with their original value since the address of the records pointed to will not change.

In our example, we have an area (PARTS) in which one CALC record resides (PARTMAST). The PARTMAST record is the owner of three sets. The member records of all sets reside in other areas. This type of structure exists in many databases.

4.2 Example One Changes

For our example, the following changes will be made:

- Area PARTS will be increased from 10,000 pages to 12,000 pages to allow for the entry of a new product line;
- The length of the CALC key of the PARTMAST record is to be increased from 21 characters to 30 characters to allow for a new part number format.

The changes will have the following effect:

- The data portion of the PARTMAST record will become three (3) words longer;
- Because of the change in area size and record key length, PARTMAST records will CALC to different pages.

Prior to the reorganization, the PARTMAST record has the following format:

```
01  PARTMAST.
    05  PART-NUMBER              PIC X(21).
    05  PART-NAME                PIC X(40).
```

After the change, it will look like this:

```
01  PARTMAST.
    05  PART-NUMBER              PIC X(30).
    05  PART-NAME                PIC X(40).
```

Note that the PART-NAME field will need to be shifted nine positions to the right to accommodate the additional characters in the key field.

In this example, the record volume is high. There are 1,000,000 REQDTL records, 100,000 PARTMAST records, 100,000 PARTXREF records, and 20,000 OPNSTEP records. Also, note that the REQDTL and OPNSTEP records have a location mode of Via Set and PARTREQ and PARTOPN sets are NOT primary sets (see page 4-1). These facts, combined with the other relationships between records in the surrounding areas, would make a conventional COBOL/DML unload and reload very difficult in terms of run time; it would take several days to unload and reload this database.

The DRU cannot be used here, since it cannot handle changes in record lengths.

Using the I-QU PLUS-1 reorganization utilities, this entire reorganization can be set up in one day or less, and executed in a few hours with minimal impact on the rest of the system. The following sections will describe how this reorganization is accomplished using I-QU PLUS-1.

4.2.1 Step-1: Collecting the Information

The first step is to modify and compile the schema and related subschemas. The new object schema and subschemas will be kept in a staging file during the reorganization so that both the old and new versions are available. Once the old and new schemas are available, it will be necessary to use QRYSCH (object schema query program) to obtain pointer position information for the PARTMAST, REQDTL, OPNSTEP and PARTXREF records. This information will be used in the coding of the I-QU PLUS-1 unload and reload programs and the set up of the PBLD and PFIX runs. The only information necessary to set up the parameters needed to perform the reorganization are the old and new schemas and the QRYSCH printouts.

4.2.2 Step-2: Building Reorganization Runs

We will need to create several runstreams to perform the reorganization:

1. Unload under the old schema and sort by CALC'ed new page number.
2. Create a NO-LOOKS schema in an alternate TIP file (SCHUTL).
3. Reload under the new schema.
4. Create a database pointer cross-reference (PBLD).
5. Relink affected areas (PFIX).
6. Verify the results.

4.2.2.1 The Area Unload

The unload process must be examined first. The unload program will be a nondestructive process as far as the current database is concerned. All the actual pointer and record changes will be made while the records are being moved to the dump tape. During the execution of unload program:

1. Each PARTMAST record in area PARTS will be retrieved using FETCH Format 3 (next of area).
2. The current database pointer (DBP) of each PARTMAST record will be retrieved from S$WORK and saved to the output record. The current DBP is the pre-reorganization record location and will be used by the I-QU PLUS-1 reload program when building the DBP cross-reference file.
3. All the set pointers for each PARTMAST record will be retrieved from the page buffer and placed in a pointer work area.
4. Empty set pointers will be nullified in the work area. This nullification process is done so that empty sets will be recognized during the reload and replaced by the new DBP of the PARTMAST record. The set pointers will then be moved from the pointer work area to the output area.
5. The data portion of the PARTMAST record will be moved to the right to make room for the new key field characters, which will be space filled.
6. The record will be pre-CALCed (CALSIM command) using the new area size and key description to determine which page it will CALC to during the reload. Records will be sorted on the resulting new page number as part of the unload process in order to minimize DMS I/O during the reload process.
7. The sorted records are then written to an output tape file.

This procedure appears to involve considerable programming, but will actually require less than 30 I-QU PLUS-1 commands to perform the entire operation.

The following is an example of the unload runstream utilizing I-QU PLUS-1:

Example One - Unload Program

```
@ASG,UP PARTS-DUMP.,T
@IQU,I
INVOKE SUBSCH-PART OF PROD-SCHEMA TIP FILE 100 FOR ST .  Old schema
DEF P H-PTRS 3                        .  Pointer work area for 3 pointers
DEF N H-CALC                          .  CALC simulation result
DEF C PARTS D 12000 1 (03010000)      .  CALC simulation key description
DEF N UNLOAD-CNT                      .  Unloaded record counter
DEF F PARTS-DUMP SEQ 92 200           .  Unload dump file
DEF A H-PART-NAME 40                  .  For moving part name
.  The following define additions to the output dump record...
DEF RDA EXPANDED-KEY (22,9)           .  New portion of CALC key
DEF RDA NEW-LOC-OF-PART-NAME (*,40)   .  New location of part name
DEF RDA NEW-CALC-PAGE (*,4) UB9       .  New CALC page number
DEF RDA SAVED-DBP (*,4) UB9           .  Old DBP save location
                                      .   (must be UB9)
DEF RDA SAVED-POINTERS (*,12)         .  Saved pointers
.

  SORT 92 92 71,4,UB9,D (40000)       .  Sort on new page number
  IMPART
  OPEN PARTS
  FETCH3 FIRST PARTS AREA
  DO WHILE ERROR-NUM = 0
    CURRENT DBP                       .  Get database ptr.  of curr.  record
    RDA SAVED-DBP = C-DBP             .  Put it in RDA for output
    GETPTR (1,3) H-PTRS               .  Get set pointers into work area
    NULPTR H-PTRS C-DBP               .  Nullify empty set pointers
    RDA SAVED-POINTERS = H-PTRS       .  Put pointer in RDA for output
    H-PART-NAME = RDA PART-NAME       .  Move name to new loc
    RDA NEW-LOC-OF-PART-NAME = H-PART-NAME
    RDA EXPANDED-KEY = ''             .  Space fill new part of key
    CALSIM PARTS H-CALC               .  Get new page number for reload
    RDA NEW-CALC-PAGE = H-CALC        .  Put CALC result in RDA
    RELEASE 92                        .  Release RDA to SORT
    UNLOAD-CNT = UNLOAD-CNT + 1
    FETCH3 NEXT PARTS AREA
  ENDDO
  DEPART
  OPEN PARTS-DUMP OUTPUT SEQ          .  Open unload dump file
.  *** Return records from sort and write to unload dump file.
  DO
    RETURN AT END BREAK               .  Return record into RDA
    WRITE PARTS-DUMP                  .  Write RDA to dump file
  ENDDO
  CLOSE PARTS-DUMP
  DISPLAY '**** Unloaded ' +
  TRIMEDIT UNLOAD-CNT 'ZZZ,ZZ9' +
  DISPLAY ' records ****'
  STOP EXIT
RUN
```

Before moving on, let us take a closer look at some of the reorganization commands used in this I-QU PLUS-1 program.

The CURRENT DBP command executed after each PARTMAST record is FETCHed retrieves the current database pointer of the record. The database pointer is delivered in the predefined variable C-DBP. C-DBP is the record's current location in the database. It will be saved with the record's data for use in re-establishing set relationships later.

The GETPTR command actually causes I-QU PLUS-1 to retrieve set pointers from the current record's control area in the DMR page buffer. In this case, three pointers, beginning with the first pointer, are retrieved. These pointers will also be saved with the record's data for use in the reload process.

The NULPTR command is used to detect and mark all empty set pointers. NULPTR only affects pointers for sets owned by the current record. I-QU PLUS-1 will scan all pointers in the pointer work area that match the value in the database pointer variable (in this case C-DBP). Each pointer that matches will be changed to a special null pointer value. This is done so that empty sets may be recognized and re-established correctly at reload time.

The CALSIM command causes I-QU PLUS-1 to call the DMSCALC routine to predetermine where the record will be stored during the reload using the new area size and key description.

The dump file will be sorted in descending order on the new page number. The page number was determined using I-QU PLUS-1's CALC simulation routine during unload. By having the records sorted in this manner, the amount of page I/O required during the reload will be greatly reduced, and the CALC chain length will be optimized.

4.2.2.2 The Area Reload

The reload process will be slightly less complicated. We simply want to reload the PARTMAST records and replace their OLD set pointers. The empty set pointers will have to be set to the new DBP of the PARTMAST records. Let us look at what will be done:

1. Each PARTMAST record will be read from the sorted dump tape.
2. The record will be stored using the I-QU PLUS-1 DML STORE command.
3. The "new" database pointer (DBP) for the record will be retrieved and saved.
4. All set pointers will be moved to a work area where empty set pointers will be changed from null to the new DBP of the PARTMAST record.
5. The set pointers will then be stored into the record in the page buffer. All pointers in the PARTMAST record will now be pointing to the original member records.
6. The database pointer cross-reference record will then be built and written.
7. This record contains the old DBP of the PARTMAST record from the dump tape and the new DBP retrieved earlier. It will be used to build a cross-reference file for relinking member records back to the PARTMAST records.
8. Steps 1 through 6 will be repeated for all PARTMAST records on the dump tape.

The entire reload process can be handled very simply using I-QU PLUS-1:

Example One - Reload Program

```
@ASG,UP DBPFILE.                        .  PTR CROSS-REFERENCE OUTPUT FILE
@ASG,A PARTS-DUMP.                      .  FROM THE UNLOAD
@IQU,I
.
INVOKE SUBSCH-PART OF PROD-SCHEMA TIP FILE 199 FOR ST .  New schema
DEF N RELOAD-CNT
DEF P H-PTRS 3
DEF F PARTS-DUMP SEQ 96 200
DEF RDA NEW-CALC-PAGE (*PART-NAME,4) UB9 .
DEF RDA SAVED-DBP (*,4) UB9             .  Saved DBP location of record from
                                        .   unload (must be UB9)
DEF RDA SAVED-POINTERS (*,12)           .  Saved set pointers from unload
  IMPART
  DBDN PARTS-ANAME = 'PARTS'            .  Database dataname for CALC
  OPEN PARTS-DUMP INPUT SEQ
  OPEN PARTS LOAD
  DO
    READ PARTS-DUMP AT END BREAK
    H-PTRS = RDA SAVED-POINTERS         .  Get pointers into work area
    X = RDA SAVED-DBP                   .  Get old DBP into variable
    STORE PARTMAST
    CURRENT DBP                         .  Get record's new DBP location
    XREF X C-DBP                        .  Write XREF -old/new location of rec
    RSTPTR H-PTRS C-DBP                 .  Restore the null set pointers
    PUTPTR (1,3) H-PTRS                 .  Put set pointers back into record.
    MODIFY PARTMAST                     .  Ensure page is written
    RELOAD-CNT = RELOAD-CNT + 1
  ENDDO
  CLOSE PARTS
  DEPART
  D '**** Reloaded ' +
  TRIMEDIT RELOAD-CNT 'ZZZ,ZZ9' +
  DISPLAY ' records.'
  STOP EXIT
RUN
```

The SET CURRENT DBP command is used after each STORE command to retrieve the record's new database pointer (or database location). It is then used in the XREF command which writes a record containing the record's old DBP (move to the variable X earlier) and its new DBP (now in C-DBP) to the DBPFILE. The DBPFILE will be used in the next run to create a database pointer cross-reference file.

The RSTPTR command scans the pointer work area, into which the set pointers have been retrieved after each STORE, for all null set pointers and replaces them with the database pointer currently in the variable C-DBP. The RSTPTR command will re-establish empty sets. The NULPTR and RSTPTR commands should only be used when the object record is an owner of sets that span areas. Also, the NULPTR and RSTPTR commands should not be used if the owner record of a set is to be PFIXed.

The PUTPTR command puts the pointers currently in the pointer work area directly into the control portion of the current record in the DMR page buffer. The I-QU PLUS-1 DML MODIFY command is used to insure that the DMR will write the page after the PUTPTR command has changed it. Remember that the DMR is not aware of the actions of the GETPTR and PUTPTR commands.

4.2.2.3 The PBLD Run

The next step is the creation of the database pointer cross-reference file. This file will be used to convert member DBPs that currently point to old locations of PARTMAST records to the new page and slot locations. This run will use program PBLD. PBLD input will be the DBP cross-reference file created by the I-QU PLUS-1 reload, and three parameters describing the schema to be used, the reorganized areas (areas in which records were moved by unloading and reloading) and the input DBP cross-reference file(s). PBLD will output a sorted and re-blocked DBP cross-reference file named DBPXREF, and create an area and database pointer parameter file called DBPPARMS. Both of these files will be passed to the PFIX runs. The schema used here must be the schema used to do the unload (the OLD schema).

Runstream to Build the DBP Cross-Reference

```
@PBLD,L
USE SCHEMA PROD-SCHEMA FILE DMS*SCHEMAFILE.
AREA PARTS                      .  Name of changed (reorganized) area
DBPFILE DBPFILE                 .  Name of input file (may be more than 1)
@EOF
```

PBLD uses the system SORT subroutine to sort database pointers internally into old database pointer sequence. The program will automatically assign temporary sort work files (XA, XB and XC), if they have not been pre-assigned to the run. If a large number of database pointers will be processed, it may be advantageous to pre-assign sort work space on separated devices or cache.

PBLD will automatically calculate the size and assign the cross-reference file as DBPXREF(+1), and the area and database pointer parameter file as DBPPARMS(+1). Both of these files will automatically be assigned as the current cycle by the PFIX runs.

4.2.2.4 The PFIX Run

The next step is the relink of member records (in other areas) back to the owner PARTMAST records. This process will use program PFIX. Input will be the cross-reference file (DBPXREF), and the area parameter file (DBPPARMS) from PBLD, and several parameters that specify:

1. Area or areas to search for records to be relinked (REQUIREMENTS, OPERATION and PARTREF);
2. Records to be examined in those areas (REQDTL, etc.);
3. Specific sets whose pointers within each record are to be examined and changed if necessary.

The information necessary to code these parameters is contained in QRYSCH program responses obtained in Step-1. Two options may be used in setting up the relink. One option is to have all relinking performed in a single run. Since the areas in our example are quite large, this option would take a considerable amount of time. The second option is to set up a separate relink run for each affected area. The separate relink runstreams will allow all surrounding areas to be relinked simultaneously. The following figure shows how records in the REQUIREMENTS, PARTREF and OPERATION areas would be linked back to the PARTS area in a single PFIX run. It would have been more efficient to perform this operation using three separate PFIX runs.

PFIX automatically assigns the input files (DBPXREF and DBPPARMS) created by PBLD, assuming PBLD was run under the same qualifier. These files may be pre-assigned and @USEed if desired.

PFIX also uses the system SORT subroutine to internally sort database pointers for matching the the DBPXREF file. Again, if large volumes are involved, pre-assigning sort work space may be advantageous.

Relink REQUIREMENTS to PARTS using PFIX

```
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH  .  Run type and schema to be used
SEARCH AREAS REQUIREMENTS,PARTREF,OPERATION
                                       .  Name of area(s) to search
RECORD REQDTL SET PARTREQ              .  Record and set affected
RECORD PARTXREF SET PARTMXREF
RECORD OPNSTEP SET PARTOPN
@EOF
```

Notice that the PARTMAST record was not PFIXed since NULPTR/RSTPTR was used in the UNLOAD/RELOAD programs.

4.2.2.5 Modifying the Schema Absolute - SCHUTL

Before running our I-QU PLUS-1 reload, we will use the SCHUTL program to make some modifications to the object schema and subschema, which substantially improve the reload run time. As these changes are destructive, they will be made to a copy of our schema file. We will need to use the original object schema and subschema after the reorganization is complete. First, area looks will be set to NOLOOKS, to cut down on DMR overhead. Second, areas will be changed to non-preinitialized so that a DMU initialize will not be necessary (assuming that areas are normally preinitialized in our schema). Last, the TIP file code in both the schema and the subschema used in the reload will be changed to 199. The new object schema and subschema will then be copied to TIP file 199 for use in the reload. The original (old) schema will still be available to other production operations involving areas not affected by this reorganization.

The runstream used to perform the object schema and subschema modifications would look like this:

```
@ASG,A TEMPSCH.
@COPY,A DMS*SCHEMAFILE.SUBPART,TEMPSCH.
@COPY,A DMS*SCHEMAFILE.PROD-SCHEMA,TEMPSCH.
@SCHUTL
USE SCHEMA PROD-SCH FILE TEMPSCH
CHANGE AREA LOOK TO NOLOOKS
CHANGE AREAS TO NON-PREINIT
CHANGE TIP FILE CODE SUBPART 199
@XQT,MXZU TIP$*TIPRUN$.TREG
TPFREE,K DMSTIPALT*SCHABSALT
@COPY,A TEMPSCH.,DMSTIPALT*SCHABSALT.
@XQT,MXQU TIP$*TIPRUN$.TREG
TPASG,K DMSTIPALT*SCHABSALT.
@FIN
```

For verification, a random set member count should be taken using a predetermined group of owner records. To accomplish this, record counts will be accumulated during the unload by a separate I-QU PLUS-1 verification run. The same verification program will be run after the reload. The set member counts produced by the "before" and "after" verification runs are compared to validate the reorganization. A DMU verify can be used here, but will run for a very long time. The spot check method may be much more practical; however, if you have any doubt, do a full DMU verify.

4.2.3 Step-3: Running the Reorganization

There are several very important points to consider when actually running the reorganization:

- A ***current back up must be taken*** of all areas involved.

- Since a single-thread DMR will be used along with non-DML page modification programs, all ***affected areas must be downed to multi-thread processing (production)***.
- There must also be some method of verifying the results of the reorganization.

To avoid the necessity of doing a save of all areas, a point immediately following a static database dump would be a good time to start the reorganization. Another possibility is to TIPOUT the areas while the non-destructive unload is being run. In either case, THE AREAS MUST BE DOWNED.

To begin the process, the unload, "before" verify, and TIPOUT are started. These runs will take less than an hour based on the record volumes given earlier.

While the unload process is running (or before), the new schema can be modified using SCHUTL and moved to the alternate TIP schema file. The production schema will not be swapped until the reorganization is complete and ready for verification.

Once the unload has been completed, a good back-up of ALL affected areas has been taken, and the "before" verification has finished, area PARTS can be recatalogued with its new size. A DMU initialize will not be necessary since the SCHUTL program was used to change areas to non-PREINITIALIZED pages.

We are now ready to run the reload. The NOLOOKS schema has been set up in the alternate TIP schema file. The reload will run quite fast, because the records were sorted on CALC'ed page number during unload, and the alternate NOLOOKS schema will be used. When this run completes, the count of records reloaded should be compared to the count of records unloaded.

Next, the run (PBLD) to build the cross reference file is executed. This run will take 5 minutes or less. As soon as this run completes, the relink run (PFIX) can be started. Using multiple relink runs (one for each area), there would have been some contention for the database pointer cross-reference file, but the overall run time will be much shorter than if using a single run as in this example. The relink run will finish in less than an hour.

At this point, all set pointers (DBPs) are pointing to the correct records. To verify that the pointers are correct, the "after" verification run should now be executed. This run will be similar to the "before" verification run.

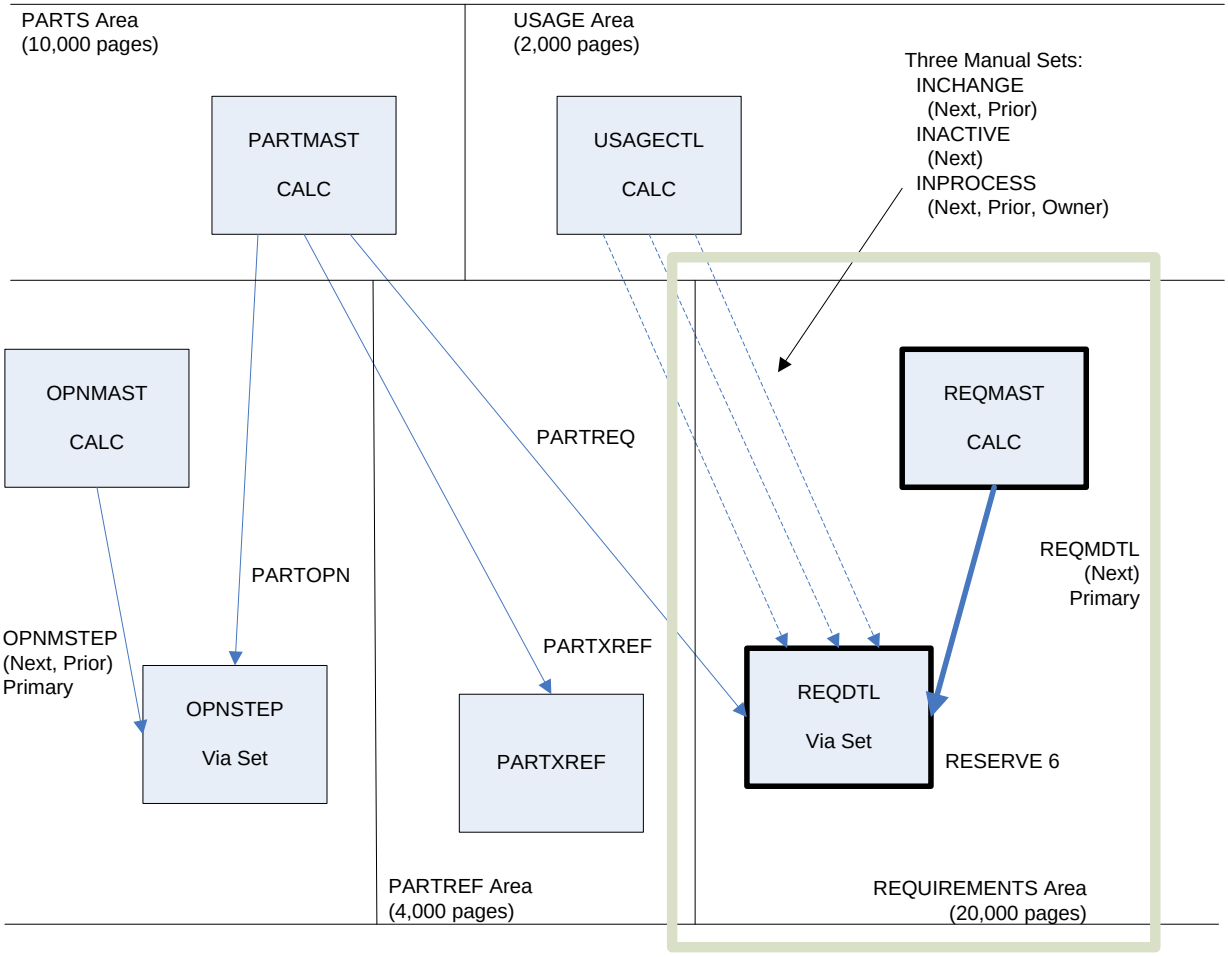
The new schema (not modified by SCHUTL) may now be placed into production. Make sure that it is also loaded into memory.

Another save of the areas should be done at this point in order to prevent problems if something unrelated to the reorganization goes wrong between the end of the reorganization and the next scheduled database backup. After the "save", we are ready to UP the areas to the multi-thread DMR.

This typical reorganization, which would not be possible using DRU, and very costly in development and run time using conventional means, was completed with less than 100 I-QU PLUS-1 commands. Although sophisticated pointer manipulation and CALC simulation were done, the high-level command language provided a reliable, proven and simple method to complete and verify the reorganization.

Chapter 5: Database Reorganization Example Two

Example 1 dealt with an area containing one record type that was an owner of several sets, all of whose members resided in other areas. In this example, the reverse situation will be considered. Here is another view of the manufacturing database example used earlier. This time, the REQUIREMENTS area will be the target of the reorganization.



5.1 The Reorganization Process

In this example, the only change to the schema will be a new 20-character field added to the end of the REQDTL record. The reorganization process will be similar to that used in the first example. The only area to be unloaded and reloaded will be the target area (REQUIREMENTS).

5.1.1 Preparations

To perform this reorganization, we need:

1. Both the old and new schema object files;
2. A QRYSCH printout for the PARTMAST, REQDTL and USAGECTL records for both the old and new schemas;
3. SCHUTL will be used to create a NOLOOKS schema in an alternate TIP schema file. Also, in order to avoid a great deal of unnecessary I/O, the SCHUTL program will be used to make one additional change to the schema. REQDTL record's participation as a member of sorted PARTREQ will be ignored using the IGNORE SET command of SCHUTL. The sort sequence of this set will be unaltered at the end of the reorganization. The SCHUTL commands are as follows:

```
@SCHUTL
USE SCHEMA PRODSCHEMA FILE TEMPSCH
CHANGE AREA LOOKS TO NOLOOKS
IGNORE SET REQDTL SET PARTREQ
```

Please note that the IGNORE SET function could not be used if the record's primary location mode is via the set to be ignored. The ignore set modifies the schema in a manner that causes the DMR to ignore the record's participation in specified set(s) when the record is stored, while the set pointer positions are preserved. This feature allows the I-QU PLUS-1 user to modify the set pointers without affecting DMR functions. You should not execute DML commands that would cause the DMR to attempt to traverse an ignored set (FETCH next of set, etc.).

5.1.2 The Unload

The unload will be fairly simple. Since the area size will not be changed, the pre-CALC and sort will not be necessary. The CALC records will generally go back to the same pages from which they were unloaded. If an excessive number of CALC records are on pages other than their prime pages (i.e., overflow), the pre-CALC and sort should be included. Each REQMAST record will be unloaded to tape followed by its member REQDTL records. Included with each REQDTL record will be its PARTREQ member pointer, manual control word and manual set pointers for the INCHANGE, INACTIVE and INPROCESS sets. These pointers will be needed to re-establish the relationship to the PARTMAST and USAGECTL records later. Also included with each REQDTL record will be its old database pointer, which is used in the cross-referencing and relink process. The REQMDTL relationship will be handled using conventional DMS 2200 techniques. Therefore, no pointer manipulation is required for either the REQMDTL set's owner or member pointers. The new 20-character field will be added to each REQDTL record, as it is unloaded.

Example Two - Unload Run

```
@ASG,UP DUMPTAPE.,T
@IQU,I
.
INVOKE SUBSCH-REQ OF PROD-SCHEMA TIP FILE 100 FOR ST
DEF P PHOLD 8                        .  Pointer work area
DEF F DUMPTAPE DBDUMP                .  Unload dump file
DEF RDA REQ-REC-DBP (1001,4) UB9     .  REQDTL database pointer
                                     .   (must be UB9)
DEF RDA REQ-REC-POINTERS (*,32)      .  REQDTL set pointers
DEF RDA REQ-REC-NEWFLD (40,20)       .  REQDTL new data field

  IMPART
  OPEN REQUIREMENTS
  OPEN DUMPTAPE OUTPUT SEQ

  F4 F REQMAST REQUIREMENTS AREA
  DO WHILE ERROR-NUM = 0
    Y = Y + 1
    WRITE DUMPTAPE REQMAST
    F3 FIRST REQMDTL SET SUPPRESS AREA
    DO WHILE ERROR-NUM = 0
      Z = Z + 1
      SET CURR DBP                   .  Get current database pointer
      GETPTR (2,8) PHOLD             .  Get all but first pointer
      RDA REQ-REC-DBP = C-DBP        .  Database pointer to RDA
      RDA REQ-REC-POINTERS = PHOLD   .  Set pointers to RDA
      RDA REQ-REC-NEWFLD=''          .  Space fill new field in RDA
      WRITE DUMPTAPE REQDTL 5 ;      .  Write REQDTL, old DBP and
      CONTROL (251,9)                .    set pointers to dump file.
      F3 NEXT REQMDTL SET SUPPRESS AREA
    ENDDO
    F4 N REQMAST REQUIREMENTS AREA
  ENDDO
  DEPART
  CLOSE DUMPTAPE
  D 'REQMAST UNLOAD COUNT = ' +
  EDIT Y 'ZZZ,ZZZ'
  D 'REQDTL UNLOAD COUNT = ' +
  EDIT Z 'ZZZ,ZZZ'
  STOP EXIT
RUN
```

5.1.3 The Reload

Before the reload is run, the temporary schema should be set up in the alternate TIP schema file (with PARTREQ ignored and NOLOOKS).

To avoid the I/O involved in reestablishing the INCHANGE, INACTIVE and INPROCESS set relationships using conventional DML methods, the reload logic will not insert REQDTL records into these sets. Instead, the pointers and manual control word saved during the unload will simply be put back into the records as they are stored. Since the DMR will be ignoring PARTREQ when the REQDTL records are stored, we will also be replacing the original PARTREQ pointers during this operation. The reload program will read the dump file and store whichever record type was read, automatically reestablishing the REQMDTL relationships with no further action required.

As each REQDTL record is stored, its old member set pointers (except REQMDTL) will be replaced, and its old and new database pointers will be used to create a database pointer, cross-reference record (XREF command).

Example Two -Reload Run

```
@ASG,A DUMPTAPE                      .  Dump file from the unload.
@ASG,UP DBPFILE.                     .  Database pointer cross-ref file
                                     .    (for PBLD)
@IQU,I
INVOKE SUBSCH-REQ OF PROD-SCHEMA TIP FILE 199 FOR ST
DEF P PHOLD 8
DEF F DUMPTAPE DBDUMP
DEF RDA REQ-REC-DBP (1001,4) UB9     .  REQDTL database pointer
                                     .    (must be UB9)
DEF RDA REQ-REC-POINTERS (*,32)      .  REQDTL set pointers
  IMPART
  OPEN REQUIREMENTS LOAD
  OPEN DUMPTAPE INPUT SEQ
  SET DBDN RQMTSAREA = 'REQUIREMENTS'
  DO
    READ DUMPTAPE AT END BREAK
    IF C-O-T = 'REQMAST'
      STORE REQMAST
      SET Y = Y + 1
    ELSE
.  *** ASSUME REQDTL RECORD READ FROM TAPE....
      PHOLD = RDA REQ-REC-POINTERS   .  Old pointers to work area
      X = RDA REQ-REC-DBP            .  Move old DBP to variable X
      STORE REQDTL
      SET CURR DBP                   .  Get new database pointer of record
      PUTPTR (2,8) PHOLD             .  Put set pointers back
      MODIFY REQDTL                  .  Insure DMR page write
      Z = Z + 1
      XREF X C-DBP                   .  Create a pointer xref. record
    ENDIF
  ENDDO
  DEPART
  DISPLAY 'REQMAST RELOAD COUNT = ' +
  EDIT Y 'ZZZ,ZZZ'
  DISPLAY 'REQDTL RELOAD COUNT = ' +
  EDIT Z 'ZZZ,ZZZ'
  STOP EXIT
RUN
```

At this point, all the records unloaded will have been reloaded. The new 20-character field has been added to the REQDTL record and initialized to spaces. The owner pointers in the PARTMAST and USAGECTL records still point to the old locations of the REQDTL records. All REQDTL records set membership pointers are pointing to either old locations of other members of their respective set or their correct original owners.

5.1.4 The PBLD and PFIX Runs

We now have all pointers for sets PARTREQ, INCHANGE, INACTIVE and INPROCESS either pointing from a REQDTL to its original owner, or pointing from an owner or member to the OLD location of a REQDTL record. To correct the pointers, the database pointer file (DBPFILE) created in the reload will be used by the PBLD program to create a database pointer cross-reference file to be used in the PFIX process. The required PBLD and PFIX runs follow:

The PBLD Run

```
@PBLD,L
USE SCHEMA PROD-SCHEMA FILE DMS*SCHEMAFILE  .  old schema
AREA REQUIREMENTS
DBPFILE DBPFILE
@EOF
```

The PFIX Runs

A. Relink pointers for PARTREQ in PARTMAST:

```
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH .New schema
SEARCH AREA PARTS
RECORD PARTMAST SET PARTREQ
@EOF
```

B. Relink pointers for INCHANGE, INACTIVE, and INPROCESS in USAGECTL:

```
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH
SEARCH AREA USAGE
RECORD USAGECTL SETS INCHANGE,INACTIVE,INPROCESS
@EOF
```

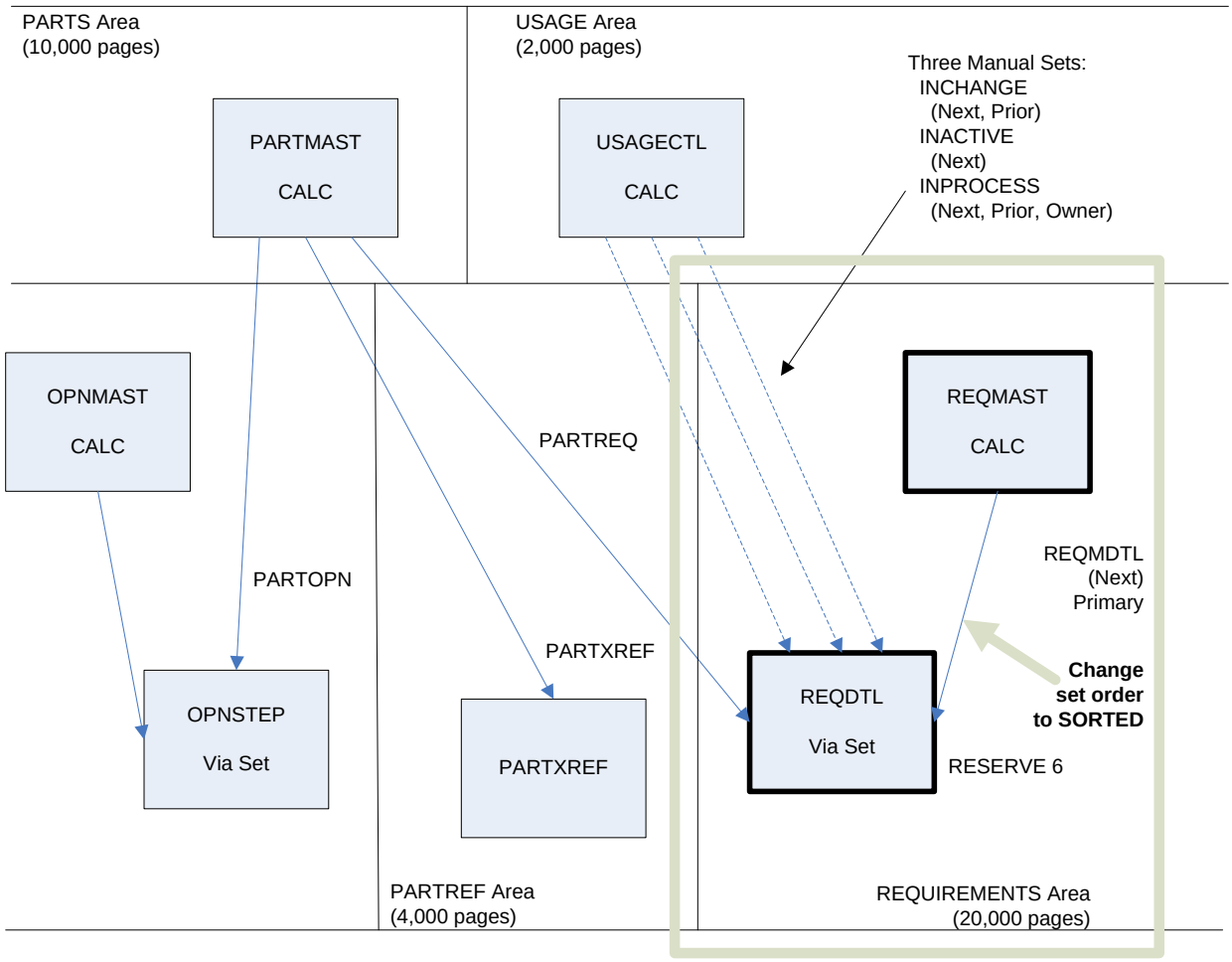
C. Relink pointers for PARTREQ, INCHANGE, INACTIVE and INPROCESS in REQDTL:

```
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH
AREA REQUIREMENTS
RECORD REQDTL SETS PARTREQ,INCHANGE,INACTIVE,INPROCESS
@EOF
```

This step completes the entire reorganization with the exception of a before and after verification, and a static save of the three areas affected. Again, this reorganization could not be done with DRU. A conventional COBOL/DML reorganization would take many times longer than the I-QU PLUS-1 method.

Chapter 6: Database Reorganization Example Three

This example will show one way in which I-QU PLUS-1 may be used to reorganize an area without doing an unload and reload. Reorganizing without unloading and reloading is only possible if the record size is not changed.



For this example, the Manufacturing Database used in the previous examples will again be used. This time, the task will be to change the order of REQMDTL from NEXT to SORTED. No other change will be made. The change in set order requires no additional pointers. It can be accomplished without physically moving records.

The process will involve one pass through the REQUIREMENTS area. Each owner REQMAST record will be fetched next of area. For each owner of set REQMDTL, we must:

1. Fetch a member REQDTL record.

2. Build a record in the RDA containing the database key, database pointer and sort key data of the REQDTL record.
3. Release the record to the SORT subroutine.
4. Repeat 1 through 3 until end of set. At end of set, continue.
5. Return a sort record and save the database key and database pointer. Fetch the current REQMAST owner, put the database pointer from the sort record into its REQMDTL pointer position and modify it.
6. Return another sort record. At end, proceed to Step 7; otherwise, save its database key and database pointer. Fetch a REQDTL record direct using the database key from the prior sort record, put the database pointer from the current sort record into its REQMDTL pointer position and modify it. Repeat until an at-end condition results from the return.
7. Fetch an REQDTL record directly using the database key from the current sort record, put the database pointer of the owner REQMAST record into its REQMDTL pointer position and modify it.

These steps are repeated until all occurrences of REQMDTL in the area are re-ordered. The process will run very fast due to the fact that the database I/Os, for the most part, will be from the beginning of the area through the end of the area (rarely will the same page be accessed twice). In addition, the multiple sorts will be done entirely in memory (in the sort buffer).

Look at the I-QU PLUS-1 program:

Sort a Set In-Place Reorganization

```
@IQU,I
INVOKE SUBSCH-REQ IN PROD-SCHEMA TIP FILE 199 FOR ST
                                  .  Use the NOLOOKS schema
DEF N OWNER-DBP                   .  Hold owner's DBP
DEF N DBP                         .  Hold database pointers
DEF N DBK1                        .  Hold database keys
DEF N DBK2
DEF P PHOLD 1                     .  Pointer work area
  IMPART
  OPEN REQUIREMENTS UPDATE
  F4 F REQMAST REQUIREMENTS AREA
  DO PROC-A UNTIL ERROR-NUM = 7
    CLOSE REQUIREMENTS
    DEPART
    DISPLAY 'REQMDTL re-ordering completed.  Processed ' +
    TRIMEDIT X 'Z,ZZZ,ZZ9' +
    DISPLAY ' occurrences of REQMDTL set'.
    STOP EXIT
.
.  Process each REQMDTL record...
PROC-A PROCEDURE
  IF OWNER REQMDTL                .  Is curr rec. owner of REQMDTL set?
    CURR DBP                      .  Get owner's DBP
    OWNER-DBP = C-DBP             .  Save owner's DBP
    SORT 20 20 1,12,DISP,A        .  Init. SORT for each set occurrence
    F3 FIRST REQMDTL SET          .  FETCH the first member
    DO PROC-SORT-RELEASE ;
      UNTIL ERROR-NUM = 7         .  Do SORT input procedure
    DO PROC-SORT-RETURN           .  Do SORT output procedure
    F2 REQMAST X=X+1              .  Count set occurrence
  ENDIF
  F4 N REQMAST REQUIREMENTS A     .  Get next REQMSTR in area
  ENDPROC                         .  Repeat, if ERROR-NUM not = 7
.  *** Build and release records to SORT.
PROC-SORT-RELEASE PROCEDURE
  CURR DBK                        .  Get current database key
  CURR DBP                        .  Get current database pointer
  RDA (13,4) UB9 = C-DBK          .  Database key to RDA (must be UB9)
  RDA (17,4) UB9 = C-DBP          .  Database pointer to RDA
```

```
                                    .    (must be UB9)
  RELEASE 20                        .  Release to SORT (key in 1-12)
  F3 N REQMDTL SET                  .  FETCH another member record
  ENDPROC                           .  Repeat, if ERROR-NUM not = 7
.
.  * Return records from SORT & replace set pointers for REQMDTL set.
PROC-SORT-RETURN PROCEDURE
  RETURN AT END PSRT-FIN            .  Should not get AT END here
  DBK1 = RDA (13,4) UB9             .  Save database key (must be UB9)
  DBP = RDA (17,4) UB9              .  Save database pointer
                                    .    (must be UB9)
  F2 REQMAST                        .  Establish currency on owner
  SETPTR (1,1) PHOLD DBP            .  Move DBP to pointer work area
  PUTPTR (1,1) PHOLD                .  DBP to REQMDTL owner ptr. pos.
  MODIFY REQMAST                    .  Insure write
  PSRT-RET
  RETURN AT END PSRT-FIN            .  Return another record
  DBK2 = RDA (13,4) UB9             .  Save database key (must be UB9)
  DBP = RDA (17,4) UB9              .  Save database pointer
                                    .    (must be UB9)
  F1 REQDTL DBK1                    .  FETCH using last SORT rec.
  SETPTR (1,1) PHOLD DBP            .  Curr. SORT rec. DBP to ptr work.
  PUTPTR (1,1) PHOLD                .  DBP to REQMDTL member ptr. pos.
  MODIFY REQDTL                     .  Insure write
  DBK1 = DBK2                       .  Rotate database keys
  GO PSRT-RET
PSRT-FIN
  F1 REQDTL DBK1                    .  Get last REQDTL
  SETPTR (1,1) PHOLD = OWNER-DBP    .  DBP of owner record
                                    .    to ptr work area
  PUTPTR  (1,1)  PHOLD              .  DBP to REQMDTL member ptr. pos.
  MODIFY  REQDTL                    .  Insure  write
ENDPROC
```

Chapter 7: Handling Pointer Array

The PFIX operation on pointer array records is different than a normal RELINK scan in that database keys (DBK), not database pointers (DBP), are being modified. This distinction is important and must be kept in mind by the reorganization programmer. The normal sequence of executing the XREF command using old and new database pointers during reload, followed by the PBLD and PFIX runs, must be expanded to include an additional XREF command execution for the database key of member records of array sets. A database pointer is composed of an area code, page number and slot offset. A database key is composed of an area code, page number and record number. However, the format of DBPs and DBKs is identical: one binary word with the same number of bits allocated for each part of the DBP or DBK. There is no characteristic of the single binary word that would allow PFIX to identify it as a DBP or DBK; therefore, database pointers and keys cannot be XREFed to the same DBP file, nor can they be relinked in the same PBLD and PFIX runs.

7.1 Review of Pointer Array Structures

Before explaining how PFIX is used to relink pointer arrays (PA) and indexed pointer arrays (IPA), a brief review of these set modes is in order. A SET MODE may be defined as CHAIN, or POINTER ARRAY. A set mode of PA may be ordered by database key or by defined keys. If the order of the PA is by defined keys, the set is an indexed pointer array.

In a chain set, the owner record points to the first member record in the set occurrence, the first member points to the next member, and so on until the last member of the set occurrence points back to the owner. In a pointer array, the owner record's next pointer does not point to the first member record. Instead, it points to a pointer array record in the associated pointer area. The array record contains the DBKs of all member records in the set occurrence. The next pointer in the member record will always point back to the owner record rather than to the next member in a PA set occurrence. Since the array record contains all member DBKs, the set occurrence is processed by the DMR by retrieving the next member DBK from the array record, and locating the member in the database using the member DBK. Although internal DMR processing is dependent upon the set mode, from an application program logical view, the set may be processed identically to a chain set. A PA set ordered by database key, will have the member DBKs in the array record ordered (sorted) by ascending database key. The format of a PA set array record is very simple: a standard record header followed by the list of member DBKs, one DBK for each member of the set occurrence, in ascending order.

An indexed pointer array (IPA) will allow the database designer to specify fields from the member record type as key criteria in ordering (sorting) the DBKs in the array. Also, an index area will be used to maintain a pointer to the highest (or lowest, if order is descending) key value from the member record stored on each page in the pointer area. As in a PA set, the next pointer of the owner record in an IPA set occurrence will point to an array record. At this point, however, the processing of the set occurrence differs from a PA.

To understand how the DMR processes an IPA set occurrence, let's first look at the pointer area and format of the IPA array records.

In a pointer area for a PA set there is one array record for each set occurrence. There will be multiple array records for an IPA set occurrence. A PA set may also have multiple array records, but only if the array record will not fit onto a single page, or if the array record has been expanded by adding new members to the set occurrence, and the expanded array record will not fit onto the page where it was originally stored. An IPA set occurrence will have one array record for each unique member key in the set occurrence. The IPA array record is composed of a record header, an index chain pointer, the DBP of the owner of the set occurrence, the concatenated key from the member record, and the DBKs of members in the set occurrence, which contains the key value. The ordering criterion for the storing of array records is using the DBP of the owner record and the concatenated key fields from the member record. The index area associated with an IPA set will contain an index entry for the highest (or lowest) owner-DBP/concatenated-member-key stored on each page of the pointer area. Since the owner-DBP is the first part of the IPA sort key, all pointer records for a single set occurrence will be logically contiguous in the pointer area. When processing an IPA set occurrence with next-of-set logic, the end-of-set condition occurs when the owner-DBP of the array record changes. The indexed pointer array structure provides random access to members within a set occurrence via the index entry for the pointer area. Fetch formats 6 and 7 are used for this type of random processing of member records.

7.2 Using PFIX With Pointer Array Sets

PFIX can read a file of XREFed database keys (DBKs) for member records for set mode of pointer array, and replace (in the pointer record) the old DBK of the member record with the new DBK of the member record in much the same way it replaced database pointers in other set modes. The RELINK scan fixing member record DBKs for PA sets is sufficient to restore database integrity. However, for IPA pointer areas, there are several additional considerations. For an IPA set, if the member record type was the object of the reorganization, and the owner record type has not been relocated in the database, relinking the member record DBKs in the array record will restore database integrity. If, however, the owner record type has been relocated during the reorganization (thus changing the owner DBP), the order of the array records in the pointer area will no longer be correct and the index entries in the associated index area will also be incorrect. If the owner record of an IPA set has been relocated, DRU must be used to rebuild the associated pointer and index areas. DRU will accomplish this reorganization task with relative efficiency.

The following rules can be applied to determine what steps are necessary to restore database integrity after affecting an IPA set by reorganization:

1. If the member record type(s) was relocated, but the owner record type was not relocated, XREF the member record DBKs during reload and PFIX the pointer area to restore database integrity. If the record's location mode is via the IPA set, the schema used during reload must have the IPA set entry in the SET SECTION changed to specify the set mode as CHAIN, or use SCHUTL's IGNORE SET command. The I-QU PLUS-1 reload program must replace the original owner pointer for the IPA set.
2. If the member record type(s) was not relocated, but the owner record type was relocated, no PFIX of the pointer area is necessary. DRU must be used to rebuild the associated index and pointer areas. In this case, the IPA set's next pointer in the owner record must have been saved at unload time (GETPTR), and replaced during reload (PUTPTR). During reload, the owner record is stored with no members, creating a null set occurrence. The PUTPTR restore of the IPA set pointer will then link the owner to the correct array record. Saving and restoring the owner pointer to the array record will allow DRU to rebuild the pointer and index areas.

3. If the member record type(s) was relocated, and the owner record type was relocated, a combination of techniques must be used in the I-QU PLUS-1 reload program to allow the set to be stored as mode chain, and then to restore the IPA pointers in the owner record (pointer to the array record) and the member record (pointer to the set owner). After the reload, perform a RELINK scan of the pointer area to PFIX the member record DBKs (as in number 1 above), and perform the DRU rebuild of the pointer and index areas (as in number 2 above). A reorganization of both the owner and member record types of an IPA set, while completely feasible using the I-QU PLUS-1 programming language and pointer manipulation commands, may be simplified by performing separate reorganizations of the owner and member record types according to the steps detailed in number 1 and 2 above.

7.3 Database Key Cross-Reference for PA/IPA

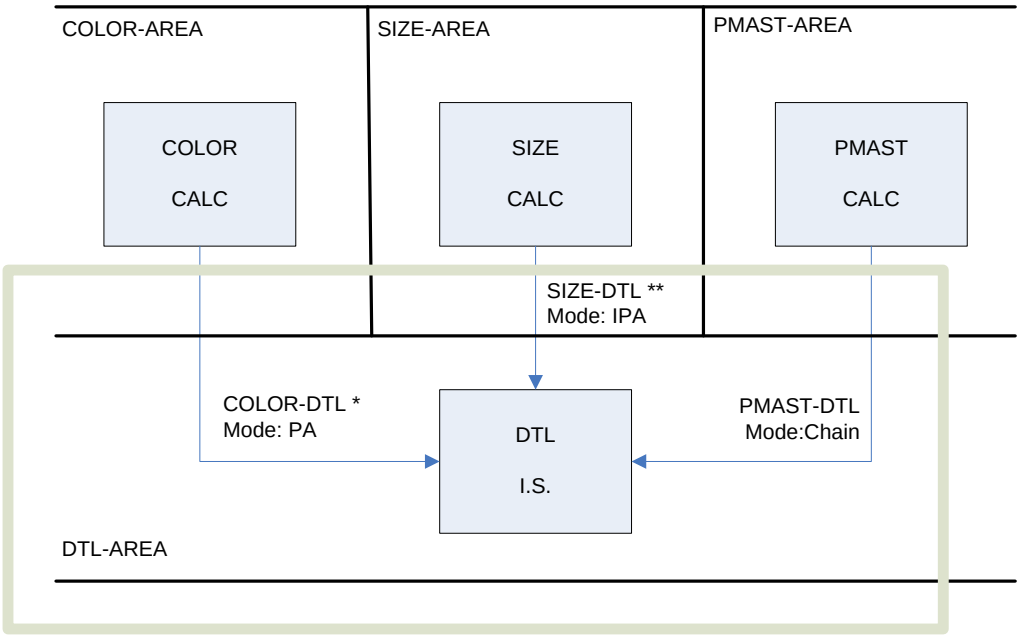
Because the DBPFILE created by the XREF command cannot contain both DBPs and DBKs, the I-QU PLUS-1 XREF command has been enhanced to allow the reorganization programmer to specify an alternate DBPFILE. The format of the XREF command follows:

```
XREF  old-ptr-or-key-var  new-ptr-or-key-var  [ALT]
```

If the ALT option is specified, the XREF record will be written to a file internally named "ALTDBPFILE". If the ALT option is not specified, the XREF record will be written to internal file name "DBPFILE". During the reorganization of IPA member records, the DBK for each member record must be XREFed to the alternate file.

7.4 Reorganizing a PA/IPA Member Area

This example performs reorganization on an area containing records that are members of a pointer array set. The procedure described here will work identically for Indexed Pointer Arrays. The following is a diagram of the areas involved:



* Pointer area for COLOR-DTL set is COLOR-PTR.

** Pointer area for SIZE-DTL set is SIZE-PTR.

The area DTL-AREA will be the target of this reorganization. The object will be to increase the DTL record size by 10 characters.

The Unload Program

```
@ASG,UP DBUNLOAD.,T
@IQU,I
INVOKE MF-2-SUB OF PROD-SCHEMA TIP FILE 100 FOR ST
DEF P SAVED-DBPS 3                .  Ptr area for saved set pointers
DEF RDA OLD-DBK (401,1) UB9       .  Output area for the old database key
                                  .    (must be UB9)
DEF RDA OLD-DBP (405,1) UB9       .  Output area for the old database ptr
                                  .    (must be UB9)
DEF RDA SET-PTRS (409,12)         .  Output area of saved set pointers
DEF F DBUNLOAD DBDUMP             .  Unload file

  IMPART
  OPEN DTL-AREA
  OPEN DBUNLOAD OUTPUT SEQ
  F4 FIRST DTL DTL-AREA A
  DO WHILE ERROR-NUM = 0
    F4 NEXT DTL DTL-AREA A
    GETPTR (1,3) SAVED-DBPS          .  Save set ptrs (2 point to
                                     .    owners) SET-PTRS = SAVED-DBPS
                                     .  Move ptrs to output area
.   *** Get the record's current database pointer...
    SET CURR DBP                     .  Retrieve DBP of member
    RDA OLD-DPB = C-DBP              .  Move DBP to output area
.   *** Get the record's current database key...
    SET CURR DBK                     .  Retrieve DBK of member
    RDA OLD-DBK = C-DBK              .  Move DBK to output area
    WRITE DBUNLOAD DTL CONTROL (100,5)
    X = X + 1
```

```
    F3 NEXT DTL DTL-AREA A
  ENDDO
  DEPART
  DISPLAY 'Unload count = ' +
  TRIMEDIT X 'ZZZ,ZZ9'
  STOP EXIT
RUN
```

Before the reload is run, an alternate TIP schema is set up using SCHUTL as follows:

```
...  copy object schema and subschema to TEMPSCH ...
@SCHUTL,L
USE SCHEMA PROD-SCHEMA FILE TEMPSCH
CHANGE AREA LOOKS TO NOLOOKS
CHANGE AREAS TO NON-PREINIT
IGNORE SET DTL SETS COLOR-DTL, SIZE-DTL, PMAST-DTL
CHANGE TIP FILE CODE MF-2-SUN 199
@.  .  .
...  copy to alternate TIP file ...
```

The Reload Program

```
@ASG,A DBUNLOAD.
@ASG,UP DBPFILE.                  .  For database pointers
@ASG,UP ALTDBPFILE.               .  For database keys
@IQU,I
INVOKE MF-2-SUB OF PROD-SCHEMA TIP FILE 199 FOR ST
DEF P SAVED-DBPS 3                .  Ptr area for saved set pointers
DEF RDA OLD-DBK (401,1) UB9       .  Output area for the old database key
.  (must be UB9)
DEF RDA OLD-DBP (405,1) UB9       .  Output area for the old database ptr
.  (must be UB9)
DEF RDA SET-PTRS (409,12)         .  Output area of saved set pointers
DEF F DBUNLOAD DBDUMP             .  Unload file

  IMP
  OPEN DTL-AREA LOAD
  OPEN DTL-INDX LOAD
  OPEN DBULOAD INPUT SEQ
  SET DBDN DTL-ANAME = 'DTL-AREA'
  DO
    READ DBUNLOAD AT END BREAK
    STORE DTL
    SAVED-DBPS = RDA SET-PTRS     .  Move saved ptrs to ptr area
    PUTPTR (1,3) SAVED-DBPS       .  Restore ptrs in member
    MODIFY MEMBER-REC             .  Ensure write to DB
.  *** Create a database pointer cross-reference record (DBPFILE)...
    SET CURR DBP                  .  Get new member DBP
    X = RDA OLD-DBP               .  Put DBP in numeric var
    XREF X C-DBP                  .  Cross Ref old/new DBP's
.  *** Create a database key cross-reference record (ALTDBPFILE)...
    SET CURR DBK                  .  Get new member DBK
    Y = RDA OLD-DBK               .  Put DBK in numeric var
    XFEF Y C-DBK ALT              .  Cross Ref old/new DBK to alternate
                                  .    DBPFILE.
    Z = Z + 1
  ENDDO
  DEPART
  DISPLAY 'Reload count = '+
  TRIMEDIT Z 'ZZZ,ZZ9'
  STOP EXIT
RUN
```

7.5 Using PBLD and PFIX on Pointer Arrays

Because there is nothing to distinguish the difference between a database pointer and a database key, a different set of PBLD and PFIX runs must be used to relink each. For example, the following runs may be used with the I-QU PLUS-1 unload/reload examples shown above.

The PBLD and PFIX run to correct database key in pointer array records:

```
@PBLD,L
USE SCHEMA PROD-SCHEMA FILE DMS*SCHEMAFILE
AREA DTL-AREA
DBPFILE ALTDBPFILE
@.
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH
AREAS COLOR-PTR, SIZE-PTR
```

No RECORD parameters are specified in the PFIX run for pointer areas.

The PBLD and PFIX run to correct database pointers in the member records:

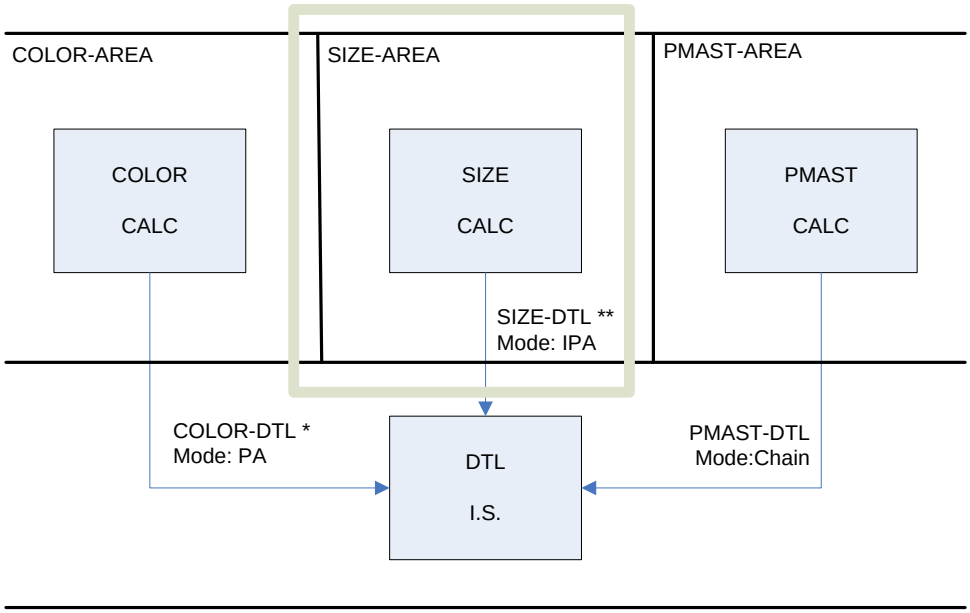
```
@PBLD,L
USE SCHEMA PROD-SCHEMA FILE DMS*SCHEMAFILE
AREA DTL-AREA
DBPFILE DBPFILE
@.
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH
AREA DTL-AREA PMAST-AREA
RECORD DTL SET PMAST-DTL
RECORD PMAST SET PMAST-DTL
```

The owner record in this case need not be fixed since it points to the pointer array record, not to the actual member record.

The pointer area has not been restructured by this reorganization. If the pointer area needs to be rebuilt (and it probably does) due to many additions and deletions, use the Database Reorganization Utility (DRU) furnished with DMS 2200. The DRU will perform the pointer area rebuild in a relatively efficient manner.

7.6 Reorganizing an IPA Owner Area

In this case, the target area will contain the owner of an indexed pointer array set. This process will work for PA as well as IPAs, with the exception being that the pointer area rebuild is not a requirement. For this example, the following database structure will be used:



* Pointer area for COLOR-DTL set is COLOR-PTR. Index area for IPA is COLORXPTR.

** Pointer area for SIZE-DTL set is SIZE-PTR. Index area for IPA is SIZEXPTR.

The target area is SIZE-AREA. The object will be to change the page size from 1792 to 448 words. The I-QU PLUS-1 programs will not need to perform any data manipulation.

The Unload Program

```
@ASG,UP  DBUNLOAD.,T
@IQU,I
INVOKE MF-2-SUB OF PROD-SCHE MA TIP FILE 100 FOR ST
DEF P PHOLD 1                          .   Ptr  area
DEF F DBUNLOAD DBDUMP                  .   Unload  file
  IMPART
  OPEN SIZE-AREA
  OPEN DBUNLOAD OUTPUT SEQ
  F4 FIRST SIZE SIZE-AREA A
  DO WHIILE ERROR-NUM = 0
    GETPTR (1,1) PHOLD                 .  Save set ptr
    SET CURR DBP                       .  Retrieve DBP of member
    RDA (1001,4) UB9 = C-DBP           .  Move DBP to output area
.  (must be UB9)
    NULPTR PHOLD C-DBP                 .  Null if empty set
    RDA (1005,4) = PHOLD               .  Move ptr to output area
    WRITE DBUNLOAD DTL CONTROL (251,2)
    X = X + 1
    F4 NEXT DTL DTL-AREA A
  ENDDO
  DEPART
  DISPLAY 'Unload count = ' +
  TRIMEDIT X 'ZZZ,ZZ9'
  STOP EXIT
RUN
```

As in other examples, the SCHUTL run will be used to modify a copy of the object schema and subschema for the reload process. In this case, we will only need to change the areas to NOLOOKS and NON-PREINITIALIZED and change the TIP file code of the schema and subschema.

The Reload Program

```
@ASG,A DBUNLOAD.
@ASG,UP DBPFILE.
@IQU,I
INVOKE MF-2-SUB OF PROD-SCHEMA TIP FILE 199 FOR ST
DEF P PHOLD 1                        .  Ptr area
DEF F DBUNLOAD DBDUMP                .  Unload file
  IMPART
  OPEN SIZE-AREA LOAD
  OPEN DBUNLOAD INPUT SEQ
  DO
    READ DBUNLOAD AT END BREAK
    PHOLD = RDA (1005,4)             .  Saved set pointer
    Y = RDA (1001,4) UB9             .  Save old DBP of rec
                                     .    (must be UB9)
    STORE SIZE SET CURRENT DBP       .  Get new DBP of rec
    RSTPTR PHOLD C-DBP               .  Fix null pointer, if empty set
    PUTPTR (1,1) PHOLD               .  Put back into SIZE rec
    MODIFY SIZE
    XREF Y C-DBP                     .  Create X-ref record
    X = X + 1
  ENDDO
  DEPART
  DISPLAY 'Reload count = ' +
  TRIMEDIT X 'ZZZ,ZZ9'
  STOP EXIT
RUN
```

At this point, there are two steps remaining before the database can be verified and returned to production. First, pointers in the member records (DTL) will have to be fixed using PBLD and PFIX. Second, the pointer area and pointer index area will have to be rebuilt. No PFIX of the pointer area is necessary. The pointer area rebuild is not necessary when the set mode is pointer array. Let us look at the PBLD and PFIX run first.

```
@PBLD,L
USE SCHEMA PROD-SCHEMA FILE DMS*SCHEMAFILE
AREA DTL-AREA
DBPFILE DBPFILE
@.
@PFIX,L
RELINK USING PROD-SCHEMA FILE TEMPSCH
AREA DTL-AREA
RECORD DTL SET SIZE-DTL
```

To rebuild the pointer area and pointer index area we will use the DRU. The following DRU run would be used in this example:

```
@DRU
IDENTIFICATION DIVISION
SCHEMA NAME IS PROD-SCHEMA IN FILE SCHEMAFILE
   SCHEMA QUALIFIER IS DMS
REORGANIZATION DIVISION
AREA SECTION
   AREA NAME IS SIZE-PTR MODIFIED UNLOADED
   AREA NAME IS SIZEXPTR MODIFIED UNLOADED
@EOF
@REORG
```

Refer to the DMS 2200 System Support Functions manual for a full description of the Database Reorganization Utility syntax.

Chapter 8: Global Database Pointer Changes

An important function of PFIX is the ability to change database area codes and/or to modify the format of database pointers (and database keys contained in pointer array records) without performing a database reorganization. Database pages are read directly from mass storage, pointers and array record database keys are modified, and the page is rewritten to mass storage. A PFIX DBPCHANGE run incurs two I/Os (read and write) for each page scanned. A DBPCHANGE run requires no internal sort operations.

The DBPCHANGE function of PFIX requires information from the current object schema and the old object schema. Area information is compared between the new and old schema to determine the affect of the change. The comparison will match areas by name; therefore, area names must not change between schemas. The only attributes that may change between old and new schemas are:

1. Area codes;
2. Area code bits changed due to change in AREA CONTROL clause in schema;
3. Page bits changed due to changes in ALLOCATE and/or EXPANDABLE TO clauses.

Any change must be reflected in the schema for it to be recognized by PFIX. For example, if data was copied from one area to another area of the same size within the same schema, PFIX could not recognize the change. PFIX is unable to recognize this change because even though the area codes in the receiving area would be incorrect, the schema definition has not changed, and PFIX is driven from the schemas.

Look at some typical examples using the DBPCHANGE function:

1. If an area's EXPANDABLE TO clause is changed from 5000 to 99999, the number of bits required for the page number will be changed from 13 to 17. If the schema is installed without changing every database pointer in the database that refers to the area, you will have an invalid database. This problem can be cured simply and quickly using PFIX's DBPCHANGE function. The PFIX run might look something like this:

   ```
   @PFIX,L
   DBPCHANGE USING PROD-SCHEMA FILE DMS*PRODSCH ;
   OLDSCHEMA PROD-SCHEMA FILE DMS*BKUPSCH
   AREAS CUSTOMERS ORDERS
   ```

   The AREAS parameter lists all areas in which database pointers that refer to the changed area will be found. If the AREAS parameter is omitted, all areas in the schema will be examined.

2. It is a common practice to copy all or part of a database to another database for use in testing and development. The problem is that very often the area codes in each database must be different. This is especially true if both use TIP areas. This application is ideal for PFIX's DBPCHANGE.

If the DBPCHANGE involves many areas or very large areas, it may be desirable to run several individual DBPCHANGE runs on different groups of areas. Use the AREAS parameter

to limit the range of each run. Make sure that all areas are covered, and make sure that no areas are scanned twice.

Chapter 9: Review of Database Record Structure

Presented here is a review of DMS 2200 database record structures. This information is covered in various DMS 2200 reference materials. It is essential that the I-QU PLUS-1 user have a thorough understanding of the location and format of set pointers within database records.

9.1 Database Pointers and Database Keys

Database keys and database pointers are single binary words containing the location of a particular record. The difference is that a database key refers to a record's area code, page number and record number, while a database pointer refers to a record's area code, page number and slot position on the page. Database pointers are used in the pointer portion of a record.

The format of database pointers and database keys is a binary word containing (left to right) an area code, page number and the slot position or record number. The number of binary bits allocated to each is variable depending upon how the schema is coded. The number of bits used for the area code will depend on the AREA CONTROL clause of the schema. The number of bits allocated in this portion of the key or pointer will be the same for all records described in the schema. The number of bits allocated will be enough to hold the maximum number of areas. For example, if the AREA CONTROL clause specifies 600 areas, the first 10 bits will be allocated to the area code in all database keys and database pointers. Following the area bits are the page bits. The number of bits allocated to the page number will depend on the ALLOCATE and EXPANDABLE TO clauses of each area described in the schema. Bits will be allocated to hold the highest possible page number. For example, if the area is expandable to 99,999 pages, 17 bits will be allocated to page number. The remaining bits will be used for record number in database keys or slot positions in database pointers. The number of page bits can vary from area to area, while the number of area bits will be consistent throughout the entire schema. The bit allocations for an area can be found in the QRYSCH report for any record that occurs within the area.

9.2 Record Structure

A physical database record consists of a record header word, followed by some number of pointers, followed by data. The number of pointers can vary from none to many depending on the record's location mode and set participation. For instance, if a record's location mode is direct and it does not participate in any sets, it will have only a record header, followed immediately by data. Otherwise, the following will apply:

Record Header (1 word)
Set Ownership Pointers present when the record owns one or more sets.
Automatic Set Membership Pointers present when the record is a member of one or more automatic sets.
Manual Control Word present when a RESERVE POINTERS clause is included in the record's definition.
Reserved Manual Set Pointers present when a RESERVE POINTERS clause is included in the record's definition.
CALC or ISAM Links
Start of Record's Data

The sequence of pointers following the record header word is as follows: owner pointers, followed by automatic member pointers, followed by the manual set control word, followed by reserved manual set pointer words, followed by CALC or indexed sequential chain pointers, followed by data.

The order of set pointers within these groups is determined by the sequence in which the sets are defined in the schema (not by set code and not alphabetically).

9.3 Physical Record Structure

9.3.1 The Record Header Word

The record header word has the following format: Starting from the left, the first 12 bits (T1) contain the record code. The next 6 bits (S4) contain record descriptor indicators. The last 18 bits (H2) contain the total record length, including the header, all pointers and data.

9.3.2 Set Ownership Pointers

Set ownership pointers will be present only if the record is defined as an owner of sets (manual or automatic). For each set there may be one or two pointers depending on the set order and links. If the set is ordered NEXT and LINKED PRIOR, two pointers will be present. Next pointers will always precede prior or last pointers. An ownership pointer will contain either the database pointer to another record, or a database pointer to itself, indicating an empty set.

9.3.3 Automatic Membership Pointers

Automatic membership pointers will be present if the record is defined as a member in one or more automatic sets. There may be from one to three pointers present for each set

depending on the set order and links. Possible pointers are next, prior and owner (in that order). A record's automatic member pointers will never point to itself.

9.3.4 The Manual Control Word and Manual Set Pointers

The manual control word and reserved manual pointers are only present if the record's definition (in the RECORD SECTION of the schema) includes a RESERVE POINTERs clause. The manual control word can be viewed as a set of 36 binary switches. Each switch may be used to indicate the record's current participation as a member of a manual set (thus the limit of 36 manual sets). The number of pointer words following the manual control word depends on the RESERVE clause of the record description, not the number of sets or set order. Use of the manual control word and pointers can best be described by example:

Assume RECORD-A is defined as a member of four manual sets with five pointers reserved as follows:

SET-W Ordered next (one pointer).

SET-X Ordered next, linked prior (two pointers).

SET-Y Ordered next, linked prior and owner (three pointers).

SET-Z Ordered last (one pointer).

The first four bits (starting from the left) of RECORD-A's manual control word will be used as switches to indicate in which of the four sets RECORD-A currently participates.

The reserved pointer words will contain only the pointers for the currently active sets. These pointers will always be packed to the left with the remaining words zero filled by the DMR.

When RECORD-A is initially stored, its manual control word and pointers will look like this:

M.C.W.	Pointer 1	Pointer 2	Pointer 3	Pointer 4	Pointer 5
0000.....	0000000000	0000000000	0000000000	0000000000	0000000000

The record's manual control word and reserved pointers will all contain zeros. If RECORD-A is then inserted into SET-W, the following changes will occur:

M.C.W.	Pointer 1	Pointer 2	Pointer 3	Pointer 4	Pointer 5
1000.....	SET-W next	0000000000	0000000000	0000000000	0000000000

The first bit of the manual control word has been turned on and reserved pointer position 1 now contains a new pointer. If RECORD-A is then inserted into SET-Y, the following changes will occur:

M.C.W.	Pointer 1	Pointer 2	Pointer 3	Pointer 4	Pointer 5
1010.....	SET-W next	SET-Y next	SET-Y prior	SET-Y owner	0000000000

The third bit of the manual control word is now on and the next three reserved pointer positions contain the next, prior and owner pointers for SET-Y. Notice that at this point, only one pointer word is left.

If an attempt is made to insert RECORD-A into SET-X, which requires two pointers, a database error will result. However, the record may still be inserted into SET-Z, which will result in the following:

M.C.W.	Pointer 1	Pointer 2	Pointer 3	Pointer 4	Pointer 5

1011.....	SET-W next	SET-Y next	SET-Y prior	SET-Y owner	SET-Z next

The fourth bit of the manual control word is now on, and the last reserved pointer now contains the next pointer for SET-Z. If RECORD-A is then removed from SET-Y, the following changes will occur:

M.C.W.	Pointer 1	Pointer 2	Pointer 3	Pointer 4	Pointer 5
1001.....	SET-W next	SET-Z next	0000000000	0000000000	0000000000

The third bit of the manual control word has been turned off, the next pointer for SET-Z has been shifted left, and the unused pointer words are zero filled by the DMR.

9.3.5 CALC and Indexed Sequential Links

These pointers are present if the record's location mode is CALC or indexed sequential. They are used to chain records to their primary page. CALC and Indexed Sequential chain pointers are not usually manipulated by the I-QU PLUS-1 user.

If you would like to help us make our documentation better, please take a few moments to complete this form and return it to KMSYS Worldwide. We are always looking for ways to improve our products and your feedback will help us reach our goal.

Name _____

Company _____

Address _____

_____

City _____ State/Province _____

Country _____ Zip/Mail Code _____

Document Name _____ OS Level _____

KMSYS Worldwide Product _____ Level _____

Please rate the documentation on a scale of 1 to 5:

	5	4	3	2	1	
Complete	❑	❑	❑	❑	❑	Incomplete
Accurate	❑	❑	❑	❑	❑	Inaccurate
Usable	❑	❑	❑	❑	❑	Unusable
Readable	❑	❑	❑	❑	❑	Unreadable
Understandable	❑	❑	❑	❑	❑	Unintelligible
Attractive	❑	❑	❑	❑	❑	Unattractive
Excellent	❑	❑	❑	❑	❑	Poor

What information did you expect to find that was omitted?

_____

Is more information needed? ❑ Yes ❑ No. If yes, on what topic?

_____

Did you find factual errors in the documentation? ❑ Yes ❑ No. If yes, please give page number and description of the error.

_____

If the documentation is difficult to understand, please specify page number and problem.

_____

Is the documentation intimidating? ❑ Yes ❑ No.

_____

Are the manuals: ❑ Too long? ❑ Too short? ❑ About the right length?

_____

Other suggestions or comments? (Use back of form if necessary.)

(Additional Comments)

. . . . . Fold along dotted line. . . . . .

P.O. Box 669695
Marietta, GA 30066
U.S.A.

Attn: Technical Documentation Section