S/36 POWER TOOLS

TIPS AND TECHNIQUES FROM NEWS 3X/400

Edited by Chuck Lundgren

DUKE PRESS

A Division of
DUKE COMMUNICATIONS INTERNATIONAL
Loveland, Colorado

DUKE COMMUNICATIONS INTERNATIONAL
Loveland, Colorado

It is the reader's responsibility to ensure procedures and techniques used from this book are accurate and appropriate for the user's installation. No warranty is implied or expressed.

This book was printed and bound in the United States of America.
First Edition: February 1991
Third Printing: January 1992

ISBN 0-9628743-0-2

About This Book

What to Do First

To receive updates for this book, along with notices of future products for the S/36 from Duke Communications, please fill out the registration card in the back of this book and mail it to:

S/36 Power Tools
Duke Communications International
PO Box 3438
Loveland, Colorado 80539 USA

What This Book Is

This book is a collection of the best tools, tips, and techniques published in the past five years in *NEWS/34-38* (pre-August 1988) and *NEWS 3X/400* (August 1988 to October 1990). This collection appeared as articles, Programs of the Month, BitStops, and Technical Corner questions and answers. You'll find more than 280 programs and procedures here, including 28 assembler subroutines.

How This Book Is Arranged

I have arranged chapters alphabetically by function group and clustered similar material within each chapter.

A cross-reference of articles and programs and procedures appears in Appendix A. The cross-reference also includes a short description of each program and procedure.

Some Caveats

Please exercise the same caution when using the procedures and programs published in this book as you would with any new routines: back up your files before using a new procedure or program with the files or when making significant changes to your files, and test all programs and procedures before placing them into production.

It is your responsibility to ensure the procedures, programs, and techniques used in this book are accurate and appropriate for your installation. No warranty is implied or expressed.

If You Encounter Problems

Every effort has been made to ensure that the programs work as the original author intended, but as with all software, there may be some anomalies (a.k.a. bugs). If you encounter problems, you can contact the editor in several ways:

(1) Mail a description of the problem to Duke Communications at the above address, or fax it to (303) 667-2321.

(2) Leave a message in the S/36 Message Base on Newslink, Duke Communications' electronic bulletin board system. For information on how to subscribe to Newslink, call (800) 373-3853 (U.S.), (800) 621-1544 (Canada), (303) 663-4700 (Colorado), or 061-976-3376 (England).
Fax (303) 667-2321, or write:

Newslink
PO Box 3438
Loveland, CO 80539 USA

How Did We Do?

We would appreciate any feedback you have on how useful this book was for you. Assuming the S/36 is around a few more years (and we have no doubts that it will be), we anticipate publishing a second volume of *S/36 Power Tools*. Your feedback will enable us to select material for that book.

How Are You Doing It Better?

If you have improved the techniques or programs published in this book, or if you have created new programs that you wish to share with the S/36 programming community, please write and tell us. We are always looking for new material for Programs of the Month, Technical Corner, and feature articles. If you just want to send us code but don't want to write the article describing it, that's okay too. If you send us the code and we accept it, we'll take care of the rest.

Interested? Send your program, an article outline, or just a query letter to:

Articles Manager
NEWS 3X/400
PO Box 3438
Loveland, CO 80539 USA

Acknowledgments

Although the cover names only one editor, actually several people wore the editor's hat. Foremost is Kathy Nelson, who not only edited and checked each published item for consistency, correctness, and clarity, but also managed the entire book project. Without her dedication, fortitude, editorial skill, creative input, and occasional harassment, this book would not exist.

Mel Beckman and Gary Kratzer provided invaluable editing and code testing for several chapters. Mel also made many suggestions that sped up and simplified the editing process. Gary and Mike Patton sifted through the material and helped determine what material constituted "best" as compared to "just okay." If any material in this book is somehow less than best, they are not to blame — the fault is mine.

The dirty jobs of digging through the diskette archives for published articles and retyping BitStop or Technical Corner pieces that had been lost and not found — as well as copying, collating, Fed Ex'ing, and Emailing — were done by the tireless Trish Frease.

Kent Rickard assembled the artwork from the previously published material.

Proofreading was done by Connie Bernard, Marie Stoupa, Nancy Arndt, John Ghrist, and Dave Bernard.

The cover concept and book design are by Steve Adams. The cover was illustrated by Bob Martin.

The layout was done by Kay Marquardt on her brand new Macintosh IIci, using Quark Xpress 3.0.

Jan Mason worked closely with the printer in scheduling and printing the book.

The book concept was Ronnie Patterson's, who also pushed for its production early on.

The authors of all the articles, Programs of the Month, BitStops, and Technical Corner pieces reprinted in this book are, in alphabetical order: Garry Abbott, Charles Ackerman, Noaman Afzal, Georgia Agallianos, Mark Allen, David Andrews, Ed Antus, George Applegate, Chuck Balsly, Robert Barber, Alex Barish, Charles Barnard, Gary Barrett, Mel Beckman, Martin Bell, George Biernadski, John Blum, Brian Blume, Mark E. Bonney, Don Bower, John B. Bowers, John Cirocco, Jeff Cole, Richard Comstock, Wells

Cooner, Steve Cranmer, H. C. Currie, B. Booth Deakins, Ray W. DeMers, William H. Dixon, Marcia Dore, Matt Drage, Ron Elliott, Teresa Elms, Eric S. Feinstein, John Field, Lou Forlini, Larry N. Forrister, Bob French, Ed Froste, John Fruetel, Perry Gardai, Tim Gardner, John Gioannetti, Ed Girou, Rick Graham, Richard Green, Robert Griffiths, Hermann Rivilla Gutierrez, Tim Hack, James H. Hamby, James Harr, E.R. Helmus, Lisa Hendricks, Matthew Henry, Bruce Hobbs, Ted Holt, Manuel Humberto, Michael Ingram, Jerry Inhoff, Sven Johnson, Deborah A. Kacerek, Debra Kahn, Gerry Karpen, John E. King, Simon Kitchen-Dunn, Barry W. Knapp, Paul Koeller, Rick Koenig, Donald J. Kott, Gary T. Kratzer, Rebecca Langren, Mark Lazarus, Darryn Lee, Steve Leichman, Alvaro de Leon, Joe Madeiros, Michael K. Maenhout, Ernie Malaga, Sarah E. McBride, Tom McLendon, Ron Mendel, George A. Meyer, Paul Michels, Judy Miller, Anthony Mossbarger, Ray Mueller, Bret B. Myrick, Sr., Abraham Notik, Michael Otey, Mike Patton, Jeffrey Pisarczyk, Timothy J. Plas, Paul Podlipny, Robert Puhalla, Heather G. Quinn, Michael J. Ranks, Esteban Riviera, Jorge Rodriquez, Bill Roehmer, Anthony Romo, Mark Rubinstein, Dennis Ruud, David C. Schlosser, Edward Schroeck, Bob Schuette, Carl W. Selley, Paul Sherrill, Nasser Shukayr, Warren Preston Sights III, Jeff Silden, Ken Sims, Bob Skowron, Grace E. Sogomian, Carson Soule, Rick Stanley, Dan Stephens, Chris Stevenson, Bruce Stradling, Thomas Straitwell, William Strejc, Burt Swan, Robin Tan, Bob Tipton, Ray Trimber, Victor J. Vergata, Nancy R. Vogelsang, Dale S. Walker, John W. Warns, Roger Washburn, Elliot Weinshenker, and Tammy A. Zitzmann.

I am very grateful for the efforts of all these people.

Chuck Lundgren
Chico, California

Table of Contents

■ = Code on diskette

Chapter 3 Data Conversion, Edits, and Validation

Chapter 4 DFU, SDA, and SEU

Chapter 5 Diskettes

Chapter 6 DisplayWrite

Chapter 7 Documentation

Chapter 8 Files

Chapter 9 Folders

Chapter 10 IDDU and Query/36

Chapter 11 Libraries

Chapter 15 Printers

Chapter 16 Programming

Chapter 17 Security

Chapter 18 System

Backup and Restore

CHAPTER

1

1

Saving and Restoring Files with Alternate Indexes

answered by Mel Beckman

Q As a consultant, I work on a wide range of customer sites, each with its own unique set of files. My problem is that I often must restore, from tape or diskette, an indexed file for which there are many alternate indexes. Sometimes the customers have saved the alternate indexes along with the file, but even in these situations, I frequently need to restore a file that now has more alternate indexes than it did when originally saved. Is there a way to rebuild all the alternate indexes after I restore the backup — without writing down the key values for each alternate index?

A A little-known fact about alternate indexes is that, when they are backed up to tape or diskette, only the key position and length information are saved; the index itself is not. When the alternate index is restored, SSP simply performs a BLDINDEX to re-create the index, using the parent file name associated with the index when it was originally saved. Thus, rather than restoring alternate indexes that may have been saved with the original file (and which won't include alternate indexes created subsequently), you should save the *existing* alternate indexes on a separate diskette.

Restoring File Groups

by Carl W. Selley

Because only the key reconstruction information is saved, alternate indexes take up practically no room on the diskette. (You can save up to 70 alternate indexes on one 2D diskette — the maximum number of datasets a 2D diskette can store.) After saving, you can safely delete the alternates, rename the original file (it is essential that you keep a copy of the original file until the backup is restored and verified), restore the backup, and restore the alternates from the diskette you just saved them on. If alternate indexes happen to use a standard dot-name prefix (e.g., CUST.X1, CUST.X2), you can use the SAVE ALL and RESTORE ALL to simplify saving and restoring the alternate indexes.

If you save more than one file group on the same tape or set of tapes or diskettes, you can save a lot of time by specifying in the S/36 SAVE procedure a set name identical to the file group prefix. For example, you would use the set name PAY for files in a group with the prefix PAY. Then when you need to restore all files within a file group, you only need specify

```
// RESTORE ALL, file-group,...
```

instead of having to restore each file individually.

Restoring a File to Disk Using a New Name

by Anthony Mossbarger

Code on diskette:
Procedure RESTFILE
Screen Format Member RESTFLFM

The RESTFILE procedure (Figure 1-1) is a tool you can use to restore diskette or tape files to disk with a different name. I have found procedure RESTFILE useful for restoring files to disk for testing or problem solving without disturbing production files. Procedure RESTFILE lets you restore all or part of a diskette or tape file.

Procedure RESTFILE uses one prompt screen (Figures1-2a and 1-2b). Two mandatory input fields, diskette or tape file name and disk file name, are entered on the prompt screen. Then, six optional input fields are available. You can indicate the number of records to be allocated to the disk file and thus limit the number of records from the diskette or tape to be restored. Or, you can specify the name of a file on disk that has the number of records needed for allocation of the disk file. If number of records to be allocated is not entered, the file from diskette or tape will be restored with its original allocation.

You can specify an input device (I1 is the default for diskettes and T1 for tape), the diskette location (S1 is the default), automatic advance to file location (default–AUTO), and you can choose to place the restore on the job queue (default–Y). If you place the restore on the job queue, several files from the same media can be restored in order. If RESTFILE is placed on the job queue, a message is sent to the originating workstation after the file is restored to disk.

All input fields are edited by procedure RESTFILE except for diskette location. If an error is detected by RESTFILE, the input screen is displayed with the appropriate message on line 24.

Figure 1-1
Procedure RESTFILE

```
******************************************************************
*  RESTFILE - THIS PROCEDURE IS USED TO RESTORE A FILE FROM      *
*             DISKETTE OR TAPE TO DISK WITH A NEW NAME           *
*                                                                *
*        PARAMETER #1 - DISKETTE OR TAPE FILE NAME (8)           *
*        PARAMETER #2 - DISK FILE NAME (8)                       *
*        PARAMETER #3 - NUMBER OF RECOROS FOR OISK FILE (8)      *
*        PARAMETER #4 - DISK FILE NAME WITH # OF RECORDS         *
*                       TO BE ALLOCATEO TO NEW FILE (8)          *
*        PARAMETER #5 - OEVICE TYPE (OISKETTE OR TAPE) (2)       *
*        PARAMETER #6 - DISKETTE LOCATION (5)                    *
*        PARAMETER #7 - AUTO ADVANCE TO FILE LOCATION (6)        *
*        PARAMETER #8 - PLACE ON JOBQ  (Y OR N) (1)              *
*        PARAMETER #9 - ERROR MESSAGE (79)                       *
*                                                                *
******************************************************************
// IF JOBQ-YES GOTO START
*
// EVALUATE P5-'I1' P6-'S1' P7-'AUTO  ' P8-'N'
```

```
// TAG AGAIN
// PROMPT MEMBER-RESTFLFM,FORMAT-SCRN01,LENGTH-'8,8,8,8,2,5,6,1,79'
// IF ?CD?/2007 CANCEL
*
// IF ?1?/ EVALUATE P9-'Enter Diskette or Tape File Name'
// IF ?1?/ GOTO AGAIN
*
// IF ?2?/ EVALUATE P9-'Enter Disk File Name'
// IF ?2?/ GOTO AGAIN
*
// IF DATAF1-?2? EVALUATE P9-'File ?2? is already on Disk'
// IF DATAF1-?2? GOTO AGAIN
*
// IFF ?4?/ IFF DATAF1-?4? EVALUATE P9-'File ?4? does not exist on Disk'
// IFF ?4?/ IFF DATAF1-?4? GOTO AGAIN
*
// IFF ?4?/ IFF ?3?/  EVALUATE P9-'Only one parameter can be entered for +
   RECORDS allocated to the Disk File'
// IFF ?4?/ IFF ?3?/ GOTO AGAIN
*
// IFF ?5?/I1 IFF ?5?/T1 EVALUATE P9-'Device for Input must be I1 or T1'
// IFF ?5?/I1 IFF ?5?/T1 GOTO AGAIN
*
// IF ?5?/I1 IFF ?7?/AUTO IFF ?7?/NOAUTO +
   EVALUATE P9-'Auto Advance must be AUTO or NOAUTO'
// IF ?5?/I1 IFF ?7?/AUTO IFF ?7?/NOAUTO GOTO AGAIN
*
// IFF ?8?/Y IFF ?8?/N EVALUATE P9-'Place on JOBQ must be "Y" or "N"'
// IFF ?8?/Y IFF ?8?/N GOTO AGAIN
*
// IF JOBQ-NO IF ?5?/I1 * 'RESTFILE ?1?,?2?,?3?,?4?,?5?,?6?,?7?,?8?'
// IF JOBQ-NO IF ?5?/T1 * 'RESTFILE ?1?,?2?,?3?,?4?,?5?'
*
// IF ?8?/Y JOBQ ?CLIB?,RESTFILE,?1?,?2?,?3?,?4?,?5?,?6?,?7?,?8?,?WS?
// IF ?8?/Y RETURN
*
// TAG START
*
// IFF ?4?/  EVALUATE P3-?F'A,?4?'?
// IF ?5'I1'?/T1 GOTO TAPE
*
// IF ?7?/AUTO EVALUATE P7-'YES'
// ELSE        EVALUATE P7-'NO '
*
*  COPY DISKETTE FILE TO DISK
// LOAD $COPY
// FILE NAME-COPYIN,UNIT-I1,LABEL-?1?,LOCATION-?6'S1'?,AUTO-?7'YES'?
// IFF ?3?/ FILE NAME-COPYO,UNIT-F1,LABEL-?2?,RECORDS-?3?
// ELSE FILE NAME-COPYO,UNIT-F1,LABEL-?2?
// RUN
// IFF ?3?/ COPYFILE OUTPUT-SAME,LIMIT-?3?
// ELSE COPYFILE OUTPUT-DISK
// END
// IF JOBQ-YES MSG ?9?,File ?1? has been copied to disk as ?2?
// RETURN
** COPY TAPE FILE TO DISK
// TAG TAPE
// LOAD $COPY
// FILE NAME-COPYIN,UNIT-T1,LABEL-?1?,VOLID-IBMIRD,RECFM-FB,RECL-256,
//      BLKL-24576,END-REWIND
// IFF ?3?/ FILE NAME-COPYO,UNIT-F1,LABEL-?2?,RECORDS-?3?
// ELSE FILE NAME-COPYO,UNIT-F1,LABEL-?2?
// RUN
// IFF ?3?/ COPYFILE OUTPUT-SAME,LIMIT-?3?
// ELSE COPYFILE OUTPUT-DISK
// END
// IF JOBQ-YES MSG ?9?,File ?1? has been copied to disk as ?2?
```

Figure 1-2a

Prompt screen RESTFILE

```
                         RESTFILE                        Optional-*

     Restores a Diskette or Tape file to disk with a new name

       Name of Diskette or Tape file.. .   .       .      ________

       Name of Disk file                     .      ..    ________

       Number of RECORDS to be allocated to the disk file ________   *

       -OR- Enter the name of a file on disk with the same
            number of RECORDS needed          . .    .    ________   *

       Enter Device for input         .   . .     I1,T1   _          *

       Diskette location          S1,S2,S3,M1.nn,M2.nn    _____      *

       Automatic advance to file location     AUTO,NOAUTO ______     *

       Place job on JOBQ ?                            Y,N _          *

                          Cmd-7 to Cancel
```

Figure 1-2b

Screen format member RESTFLFM

```
*        1         2         3         4         5         6         7         8
0001 SSCRN01
0002 D            8 136Y                  Y                   CRESTFILE
0003 D           10 169Y                                      COptional-*
0004 D           56 3 9Y                                      CRestores a Diskette or X
0005 DTape file to disk with a new name
0006 D           51 511Y                                      CName of Diskette or TapX
0007 De file
0008 DI1NAME      8 567Y  Y     Y         Y         Y
0009 D           51 711Y                                      CName of Disk file      X
0010 D
0011 DF1NAME      8 767Y  Y               Y         Y
0012 D           51 911Y                                      CNumber of RECORDS to beX
0013 D allocated to the disk file
0014 DRECIN       8 967Y  YN              Y         Y
0015 D            1 978Y                                      C*
0016 D           511111Y                                      C-OR- Enter the name of X
0017 Da file on disk with the same
0018 D           461216Y                                      Cnumber of RECORDS neededX
0019 Dd
0020 DFILE        81267Y  YB              Y         Y
0021 D            11278Y                                      C*
0022 D           511411Y                                      CEnter Device for input.X
0023 D                         I1,T1
0024 DDEVICE      21467Y  YB              Y         Y
0025 D            11478Y                                      C*
0026 D           511611Y                                      CDiskette location      X
0027 D         S1,S2,S3,M1.nn,M2.nn
0028 DDLOC        51667Y  Y               Y         Y
0029 D            11678Y                                      C*
0030 D           511811Y                                      CAutomatic advance to fiX
0031 Dle location       AUTO,NOAUTO
0032 DAUTO        61867Y  YA              Y         Y
0033 D            11878Y                                      C*
0034 D           512011Y                                      CPlace job on JOBQ ?    X
0035 D                           Y,N
0036 DJOBQ        12067Y  YA          Y   Y         Y
0037 D            12078Y                                      C*
0038 D           152331Y                                      CCmd-7 to Cancel
0039 DERRMSG      7924 2Y  Y              Y Y
```

Saving All User Libraries

program by David Andrews

Code on diskette:
Procedure LIBBAK
RPG program LIBBAK
Screen format member LIBBAKFM
Message member LIBMSG

Utility LIBBAK lets you save all libraries in one step instead of saving them one at a time with the SAVELIBR command.

Regularly backing up your S/36 files and libraries to diskette is essential for your archives and for recovery in case of accidental data loss. Saving your files is quick and easy because you can use the SAVE ALL command to back up all your files at once; saving your libraries isn't so simple. The misnamed SAVE ALL command won't save all your libraries at once. Although the SAVELIBR command commonly is used to back up libraries, you must supply the library name as one of SAVELIBR's parameters — which means you can back up only one library at a time. If you have a lot of libraries to save, backing them up individually can be a lengthy process.

Utility LIBBAK lets you back up all your user libraries to diskette in one step. LIBBAK also lets you save individual libraries. Utility LIBBAK comprises RPG program LIBBAK, screen format member LIBBAKFM, message member LIBMSG, and procedure LIBBAK.

Figure 1-3
LIBBAK prompt screen

```
                              LIBBAK PROCEDURE

           Saves a specified library or ALL libraries
           to diskette

   Name of library to save or ALL                     .       .     ALL

   Volume ID of diskette(s)  .            .       .               BACKUP

   Beginning diskette location    . . .      S1,S2,S3,M1.nn,M2.nn     S1


   Cmd3-Previous Menu       Cmd7-End
```

To execute the utility, type LIBBAK. A prompt screen (Figure 1-3; screen format member LIBBAKFM is in Figure 1-4) displays three default parameters. The first parameter is either the name of an individual library you want to save or the useful default ALL. Parameter 2 is the backup diskette's volume ID, which defaults to BACKUP, and parameter 3 is the beginning diskette slot location, which defaults to S1. (You can change the procedure to specify as defaults for parameters 2 and 3 the volume ID and slot location you most commonly use.) Parameters 2 and 3 need to be entered only once.

Procedure LIBBAK (Figure 1-5) performs an ALLOCATE that ensures you have dedicated use of the diskette drive. LIBBAK also uses the

AUTO-YES,CONTINUE-YES keywords to locate the next available diskette slot location automatically while you save the libraries.

To save an individual library, specify a library name in parameter 1, enter values for parameters 2 and 3 if necessary, and press Enter. Procedure LIBBAK verifies the library's existence and the validity of parameters 2 and 3 before continuing. If procedure LIBBAK detects an error, the screen is redisplayed with the questionable field in reverse image and the error message at the bottom of the screen (see Figure 1-6 for message member LIBMSG). To continue, correct the error and press Enter. LIBBAK then saves the library (using SAVELIBR) and redisplays the prompt screen. Enter the name of the next library you want to save, or press Command key 7 to end LIBBAK.

To save all libraries, accept the parameter 1 default ALL, specify parameters 2 and 3 if necessary, and press Enter. Before utility LIBBAK saves all libraries, it creates a file in a format that can be converted into a save procedure. First, $LABEL generates a VTOC list that is saved in disk file SAVEPRNT. Then #GSORT sorts file SAVEPRNT in library name sequence and outputs file SAVEPRT2. An alternate index named SAVEPRTX is built to provide a key (consisting of the eight-character library names) over file SAVEPRT2 so program LIBBAK can read the file multiple times.

Program LIBBAK (Figure 1-7) is loaded to read file SAVEPRT2 and to output, in $MAINT format, disk file LIBBAK, which contains the OCL necessary to save all your user libraries in SAVELIBR format. (#LIBRARY is not considered a user library, so LIBBAK will not save it.) Finally, $MAINT copies the file into the current library to create procedure LIBBAK?WS?, which is called to perform the actual backup. After the libraries are saved, LIBBAK performs housekeeping that deletes the LIBBAK?WS? procedure and any remaining work files.

Utility LIBBAK lets you save all your libraries to diskette in one easy step. With some simple modifications, LIBBAK also can back up libraries onto tapes and save #LIBRARY. So next time you run a SAVE ALL — which saves "almost all" — run utility LIBBAK to make your backup complete.

Figure 1-4

Screen format member LIBBAKFM

```
*        1         2          3    . . 4       . 5        . 6  .  .  7 .     8
SPROMPT    0124    29YY    29                                  CDG
D#$CONS    16 133Y                                       CLIBBAK PROCEDURE
D#$CONS    42 320Y                                       CSaves a specified libraX
Dry or ALL libraries
D#$CONS    11 420Y                                       Cto diskette
D#$CONS    63 6 4Y                                       CName of library to saveX
D or ALL                                    .   .
DLIBRARY    8 66901 Y     21           21  Y
D#$CONS    63 8 4Y                                       CVolume ID of diskette(sX
D)                                        .
DVOLUMEID   8 86902 Y     22           22  Y
D#$CONS    6310 4Y                                       CBeginning diskette locaX
Dtion                 S1,S2,S3,M1.nn,M2.nn
DLOCATION   5106903 Y     23           23  Y
D#$CONS    7523 2Y                                       CCmd3-Previous Menu    CX
Dmd7-End
```

Figure 1-5

Procedure LIBBAK

```
0017 DMESSAGE   7524 229              29    29          M
**************************************************
**  Set up initial procedure attributes         **
**************************************************
// MEMBER USER1-LIBMSG
**
**************************************************
**  Procedure may only be run on the console    **
**************************************************
// IF CONSOLE-NO  ERR 0004,23
// IF CONSOLE-NO  RETURN
**
**************************************************
**  Allocate diskette drive                     **
**************************************************
// TAG ALLOC
// ALLOCATE UNIT-I1,AUTO-YES,CONTINUE-YES,WAIT-NO
// IF ?CD?-2032  ERR 0001,123
// IF ?CD?-2033  ERR 0001,123
// IF ?CD?-1011  GOTO ALLOC
// IF ?CD?-1012  RETURN
**
**************************************************
**  Test for one-time use                       **
**************************************************
// SWITCH 0XXXXXXX
// IFF ?1?- IFF ?2?- IFF ?3?-   SWITCH 1XXXXXXX
// IFF ?1?- IFF ?2?- IFF ?3?-   GOTO SKIP
**
**************************************************
**  Processing for prompt screen.               **
**************************************************
// IF ?1?-  EVALUATE P1-ALL
// IF ?2?-  EVALUATE P2-BACKUP
// IF ?3?-  EVALUATE P3-S1
// TAG PROMPT
// PROMPT MEMBER-LIBBAKFM,FORMAT-PROMPT,START-1,LENGTH-'8,8,5,0,0,0,0,0,0,6'
// IF ?CD?-2007  DEALLOC UNIT-I1
// IF ?CD?-2007  RETURN
// IF ?CD?-2003  DEALLOC UNIT-I1
// IF ?CD?-2003  RETURN
// IF ?1?-       GOTO PROMPT
**
**************************************************
**  Screen error processing.                    **
**************************************************
// TAG SKIP
// EVALUATE P10-'' P21-'' P23-'' P23-'' P29-''
// LOCAL OFFSET-1,DATA-'?3?',BLANK-5
// IFF ?L'1,2'?-S1 IFF ?L'1,2'?-S2 IFF ?L'1,2'?-S3 IFF ?L 1,2'?-M1 IFF ?L'1,2'?-M2 +
                                  EVALUATE P23-1 P29-1 P10-0002U1
// IF ?L'1,1'?-M IFF ?L'3,1'?-    EVALUATE P23-1 P29-1 P10-0002U1
// IF ?L'1,1'?-M IFF ?L'4,2'?>00  EVALUATE P23-1 P29-1 P10-0002U1
// IF ?L'1,1'?-M IF ?L'4,2'?>10   EVALUATE P23-1 P29-1 P10-0002U1
// IF ?L'1,1'?-S IFF ?L'3,3'?-    EVALUATE P23-1 P29-1 P10-0002U1
// IF ?29?-1                      GOTO PROMPT
// IFF ?1?-ALL IFF DATAF1-?1?     EVALUATE P21-1 P29-1 P10-0003U1
// IF ?29?-1                      GOTO PROMPT
// IFF ?VOLID'?3?'?-?2?           EVALUATE P22-1 P29-1 P10-0005U1
// IF ?29?-1                      GOTO PROMPT
**
**************************************************
**  Save a single library                       **
**************************************************
// IF ?1?-ALL     GOTO SAVEALL
// * 'Library ?1? is now being saved to diskette'
// SAVELIBR ?1?,1,?2?,?3?
// IFF SWITCH1-1  GOTO PROMPT
// IFT SWITCH1-1  DEALLOC UNIT-I1
// IFT SWITCH1-1  RETURN
**
**************************************************
**  Save all libraries                          **
**************************************************
```

```
// TAG SAVEALL
// * 'Saving ALL libraries to diskette'
*
* Delete workfiles
// IF DATAF1-SAVEPRNT  DELETE SAVEPRNT,F1,REMOVE
// IF DATAF1-SAVEPRTX  DELETE SAVEPRTX,F1,REMOVE
// IF DATAF1-SAVEPRT2  DELETE SAVEPRT2,F1,REMOVE
// IF DATAF1-LIBBAK    DELETE LIBBAK,F1,REMOVE
*
* Generate VTOC file
// LOAD $LABEL
// RUN
// DISPLAY UNIT-F1,LABEL-ALL,SORT-NAME,OUTPUT-SAVEPRNT
// END
*
* Select all of the libraries from the VTOC listing
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-SAVEPRNT,RETAIN-S
// FILE NAME-OUTPUT,LABEL-SAVEPRT2,RECORDS-?F'A,SAVEPRNT'?,EXTEND-10,RETAIN-T
// RUN
     HSORTR     8A        3X   8   N
     I C  26  32EQCLIBRARY
     FDC   1   8                        LIBRARY NAME
// END
*
* End procedure if no libraries exist
// IF ?F'A,SAVEPRT2'?=0  DEALLOC UNIT-I1
// IF ?F'A,SAVEPRT2'?=0  ERR 0006,23
// IF ?CD?=1012  RETURN
*
// LOCAL OFFSET-11,DATA-'?2?',BLANK-6
// LOCAL OFFSET-21,DATA-'?3?',BLANK-5
// LOCAL OFFSET-31,DATA-'?WS?',BLANK-2
// EVALUATE P64=?F'A,SAVEPRT2'?*2+5
// BLDINDEX SAVEPRTX,1,8,SAVEPRT2
*
* Create file that will later be converted to procedure
// LOAD LIBBAK
// FILE NAME-SAVEPRNT,LABEL-SAVEPRTX
// FILE NAME-LIBBAK,RECORDS-?64?,EXTEND-10,RETAIN-T
// RUN
*
* Change file into executable procedure
// IF PROC-'LIBBAK?WS?,?CLIB?'  REMOVE LIBBAK?WS?,PROC,?CLIB?
// LOAD $MAINT
// FILE NAME-LIBBAK,RETAIN-S
// RUN
// COPY FROM-DISK,TO-?CLIB?,FILE-LIBBAK,NAME-LIBBAK?WS?
// END
*
* Save libraries, deallocate diskette drive, perform REMOVEs/DELETEs
// INCLUDE LIBBAK?WS?
// DEALLOC UNIT-I1
// REMOVE LIBBAK?WS?,PROC,?CLIB?
// DELETE SAVEPRTX,F1,REMOVE
// DELETE SAVEPRT2,F1,REMOVE
**
// RETURN
**
*****************************************************************
**  PROCEDURE    LIBBAK                                        **
**                                                             **
**  WRITTEN BY   Dave Andrews                                  **
**                                                             **
**  DESCRIPTION  This procedure saves all of the libraries     **
**               on the system or a selected library to        **
**               diskette                                      **
**                                                             **
**  PARAMETERS   ?01? = Name of library to save or ALL         **
**               ?02? = Diskette volume ID                     **
**               ?03? = Beginning diskette location            **
**               ?21? = Position cursor/reverse image on screen **
**                      for invalid library name error         **
**               ?22? = Position cursor/reverse image on screen **
**                      for invalid volume ID error            **
```

```
**            ?23? - Position cursor/reverse image on screen **
**                   for invalid diskette location error     **
**            ?29? - Sound alarm/display error message on    **
**                   prompt screen                           **
***************************************************************
```

Figure 1-6

Message member LIBMSG

```
LIBMSG,1
0001  Diskette drive is not available now
0002  Invalid diskette location
0003  Specified library does not exist on the disk
0004  Procedure can only be run on the system console
0005  Specified volume id does not match with diskette volume id
0006  No libraries are on the system to save
```

Figure 1-7

Program LIBBAK

```
*. .   . 1      . 2 ... ... 3     .. 4 .. .   5 .     6 ..  .. 7 .      8
0001 H    P064                     B      1                       LIBBAK
0002 H/SPACE
0003 H*********************************************************
0004 H**  PROGRAM      LIBBAK                                **
0005 H**                                                     **
0006 H**  WRITTEN BY   Dave Andrews                          **
0007 H**                                                     **
0008 H**  DESCRIPTION· This program creates a file containing all  **
0009 H**               of the libraries to be backed up onto       **
0010 H**               diskette in a format that can be converted  **
0011 H**               into a procedure.                           **
0012 H*********************************************************
0013 FSAVEPRNTIF  F 256   8L 8AI      1 DISK                      LIBRARY FILE
0014 FLIBBAK  O   F 120 120           DISK               A        PROCEDURE FILE
0015 E                    CNST    1   5 19             CONSTANTS FOR OUT ARRAY
0016 E                    OUT       120  1             ARRAY FOR OUTPUT LINE
0017 ISAVEPRNTID  01
0018 I                                      1   8 LIBNAM          LIBRARY NAME
0019 I/SPACE 2
0020 I            UDS
0021 I                                     11  16 VOLID           VOLUME ID OF DISKETTES
0022 I                                     21  22 LOCATN          DISKETTE LOCATION
0023 I                                     31  32 WSID            WORKSTATION ID
0024 C*********************************************************
0025 C**          PROGRAM OUTLINE                              **
0026 C*********************************************************
0027 C                     MOVE CNST,1    CNST1  19
0028 C                     MOVE CNST,2    CNST2  19
0029 C                     MOVE CNST,3    CNST3  19
0030 C                     MOVE CNST,4    CNST4  19
0031 C                     MOVE CNST,5    CNST5  19
0032 C                     MOVE *BLANKS   KEY     8
0033 C                     EXCPTSTART
0034 C**
0035 C            KEY      SETLLSAVEPRNT
0036 C                     MOVE 'N'       EOF     1
0037 C            EOF      DOUEQ'Y'
0038 C                     READ SAVEPRNT                50
0039 C          50         MOVE 'Y'       EOF
0040 C            EOF      IFNE 'Y'
0041 C                     MOVE *BLANKS   OUT
0042 C                     MOVEACNST1     OUT,1
0043 C                     MOVEALIBNAM    OUT,14
0044 C                     EXSR $FIND
0045 C                     MOVEACNST2     OUT,B
0046 C                     ADD  14        B
0047 C                     MOVEAVOLID     OUT,B
0048 C                     EXSR $FIND
0049 C                     MOVEACNST3     OUT,B
0050 C                     ADD  19        B
```

```
C                     MOVEALOCATN    OUT,B
C                     EXCPTDETAIL
C                     END
C                     END
C**
C                     EXCPTRUN
C**
C           KEY       SETLLSAVEPRNT
C                     MOVE 'N'       EOF     1
C           EOF       DOUEQ'Y'
C                     READ SAVEPRNT                 50
C         50          MOVE 'Y'       EOF
C           EOF       IFNE 'Y'
C                     MOVE *BLANKS   OUT
C                     MOVEACNST4     OUT,1
C                     MOVEALIBNAM    OUT,18
C                     EXSR $FIND
C                     MOVEACNST5     OUT,B
C                     ADD  14        B
C                     MOVEALIBNAM    OUT,B
C                     EXCPTDETAIL
C                     END
C                     END
C**
C                     EXCPTEND
C                     SETON                     LR
C/SPACE 2
C****************************************************************
C**         $FIND  - FIND THE END OF THE CHARACTER STRING     **
C****************************************************************
C           $FIND     BEGSR
C                     Z-ADD120       A       30
C                     Z-ADD0         B       30
C           A         DOUEQ0
C           OUT,A     IFNE *BLANK
C                     Z-ADDA         B
C                     Z-ADD0         A
C                     ELSE
C                     SUB  1         A
C                     END
C                     END
C                     ADD  1         B
C                     ENDSR
OLIBBAK  EADD             START
O                                   23 '// COPY LIBRARY-P,NAME-'
O                                   29 'LIBBAK'
O                         WSID      31
O        EADD             START
O                                   14 '// LOAD $MAINT'
O**
O        EADD             DETAIL
O                         OUT      120
O**
O        EADD             RUN
O                                    6 '// RUN'
O**
O        EADD             END
O                                    6 '// END'
O        EADD             END
O                                    7 '// CEND'
**  CNST
// FILE NAME-
,UNIT-I1,PACK-
,RETAIN-1,LOCATION-
// COPYLIBR FROM-
,TO-DISK,FILE-
```

Saving #LIBRARY to Tape

answered by Mel Beckman

Q Whenever we try to back up to tape, we get this message:

```
LIBSVALL
SAVELIBR PROCEDURE IS RUNNING
SYS-2401 OPTIONS ( 123 )
CANNOT SAVE THE SYSTEM LIBRARY ON TAPE...
```

Is the ability to perform this backup for the system library new to Release 5.0? Or is there something in our configuration (Model B24 5360 with a nine-track tape unit and three 200 MB disk drives) that does not permit us to perform this backup? We now use SSP Release 4.0.

A Yes, the ability to back up the system library to tape is new to Release 5.0. Restoring #LIBRARY from tape is also supported in this expanded function. One way to complete the reload from tape is to specify:

```
IPL TC
```

or

```
IPL T1
```

Another way is to mount the #LIBRARY tape and then perform an IPL from diskette. (The procedure for this is different for each machine and is described in IBM's manual *Operating Your Computer.*) When you IPL from diskette, the system first checks to see whether a tape is mounted; if so, the IPL takes place from the tape drive.

There is nothing in your configuration to prevent Release 5.0 from performing this backup.

Finding the Number of Active Jobs

by Mel Beckman

Code on diskette:
Assembler program ACTIVE

Suppose the COMPRESS step in your S/36 nightly batch job encounters a spool writer or a MRT job. If the COMPRESS senses that the system is not dedicated, it issues an operator message. Unfortunately, no one is there to answer the message, and the entire nightly job hangs up. What you need in your nightly batch jobs is a procedure substitution expression that returns the number of active jobs so the batch job can test whether the system is dedicated. If the system is not dedicated, the COMPRESS step can be bypassed temporarily.

IBM neglected to supply such a procedure substitution expression, but assembler program ACTIVE returns the number of currently active jobs via the ?CD? substitution parameter.

The resulting assembler program returns the job count in the substitutional parameter ?CD?. This count includes spool writers and other system jobs that affect dedication status, but does not include communication tasks and command processors that do not compromise dedicated mode. Because the program returns the number of active jobs through a substitutional parameter, you can use this program with any procedure without regard to how the procedure uses UPSI switches or the LDA.

The sample OCL in Figure 1-8 will keep the nightly batch job from stalling in the COMPRESS step. Note how the COMPRESS command is executed only if a single job is running (you must condition on a job count of one because there is always at least one job running).

Figure 1-8

Sample OCL code that checks for active jobs

```
*
* Don't attempt COMPRESS unless the system is dedicated
*
// LOAD ACTIVE
// RUN
// IF ?CD?/0001 COMPRESS
```

Re-creating Program Active

If you don't have assembler routine ACTIVE, you can re-create it with procedure MKACTIVE. (You don't need IBM's Assembler Language Program Product to install ACTIVE.) To run MKACTIVE, you must be signed on as a security officer, and the system must be dedicated.

```
** Use $MAINT to copy #CNHLP to ACTIVE so we have a load member to patch **
// LOAD $MAINT
// RUN
// COPY FROM-#LIBRARY,TO-#LIBRARY,LIBRARY-O,NAME-#CNHLP,NEWNAME-ACTIVE,RETAIN-P
// END
**
** Patch the new ACTIVE member to make it set file delete flags
**
// LOAD $PFIX
// RUN
MOD
PTF OACTIVE,,#LIBRARY
DATA F2,0000,3544,086A,0830,1896,1832,3744,1838,F101,00
DATA 00,0010,00F4,000F,8A26,0075,A117,4801,8A18,3348,03
DATA 03,0020,0384,1634,4901,8818,3548,0388,1538,F400,04
DATA 04,0030,0404,F1F0,F0F0,F000,01F0
END
```

Backing Up at Night or in the Morning

answered by Mel Beckman

Q I would like some feedback on the age-old debate about whether it's best to do your daily system backup first thing in the morning or at the end of the day in a one-shift, one-programmer shop.

A Evening backup is safer if your company operates primarily during the daytime. If you wait until the morning, an entire day's work exists solely in the machine for 12 or more hours, and thus your data is vulnerable *more than half the time* to lightning, fire, flood, and criminal assault. The extra effort that an evening backup requires pays off in acceptable protection.

The Order in Which Files Are Saved to Tape

answered by John Fruetel and Burt Swan

Q What is the order of the file backup when I do a SAVE ALL? We currently back up onto 16 to 18 tapes, and finding the files is a long task.

A The system saves files in VTOC (random) order, but alternate indexes appear last because they contain no actual "data." With alternate indexes, the SAVE provides a record that describes how to rebuild the index, so it is important that the indexes be restored *after* the actual data files.

To gain control of the SAVE order, you can rename the application files with a group name (e.g., AP.*xxxxx* for accounts payable files) and then save each group to its own set name:

```
SAVE ALL,1,group name,volid,set name,T1,LEAVE
```

For simplicity, make your group names and set names the same. After saving file groups, do a separate SAVE of all files that do not belong to a group.

Assuming your file sizes are not too volatile, you should have a good idea of the reel on which a file set begins after cataloging a set of backup tapes. Using 3,600-foot tapes can reduce the number of reels by 50 percent, correspondingly reducing your number of choices. If you have historic data on disk that you do not need to back up every day, you also can reduce your choices by giving these files group names but *not* saving them. This practice keeps the historic data from taking up space on the daily backup tapes.

If you need to restore by application, this method is quite convenient:

```
RESTORE ALL,set name,,T1
```

If you are trying to restore only one file, the method admittedly may be cumbersome.

SAVE/COMPRESS Algorithm

Answered by Mel Beckman

Q I need some information about how the SAVE/COMPRESS algorithm works. In our company, we need to transfer information from tape to microfiche, but none of the local microfiche companies can handle compressed data. If I can get answers about the compression algorithm, the microfiche companies can create some special programs to convert the data. Without the special programs, we must make about 200 diskettes each time we store data. This process takes considerably more time than we want to expend. Please help us; IBM won't!

A The S/36 cannot generate compressed data to an attachable tape drive. Because this restriction stems from how the S/36 Control Storage Processor microcode has been written, you cannot override it with a simple OCL change. However, many (if not most) computer output microfiche companies can accept diskettes in lieu of tapes.

But, to answer your question — the S/36 SAVE/COMPRESS algorithm is really quite straightforward. The option to save data files in a compressed format is controlled by a parameter in the SAVE procedures and $COPY utility program. (Library and folder members are already stored in a compressed format.) If you select this parameter, redundant and repetitive characters are removed as you copy the data to diskette. These characters are replaced by control characters, which allow reconstruction to the data's original format on disk.

The incoming data file can be defined in terms of three different string types: nonduplicate (the string contains no duplicate characters), prime duplicate (the string contains only characters of prime value, the implemented prime value being that of the blank, X'40'), and nonprime duplicate (the string contains consecutive identical nonblank characters). Each of these string types in the original data file is converted to a compressed string (Figure 1-9).

The first byte on a compressed string is a control byte that defines the string type and the string length. In Technical Bulletin G360-1009, IBM provides additional information about control bytes as they relate to string types. The construction of the control byte is illustrated in Figure 1-10.

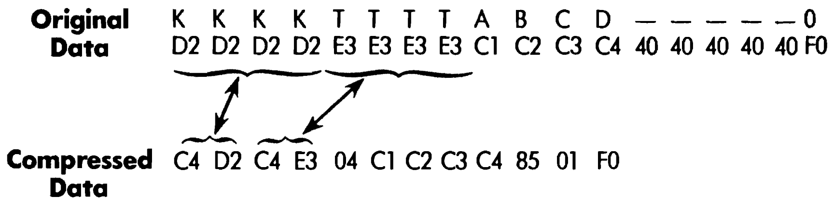


Figure 1-9
Example of compressed data

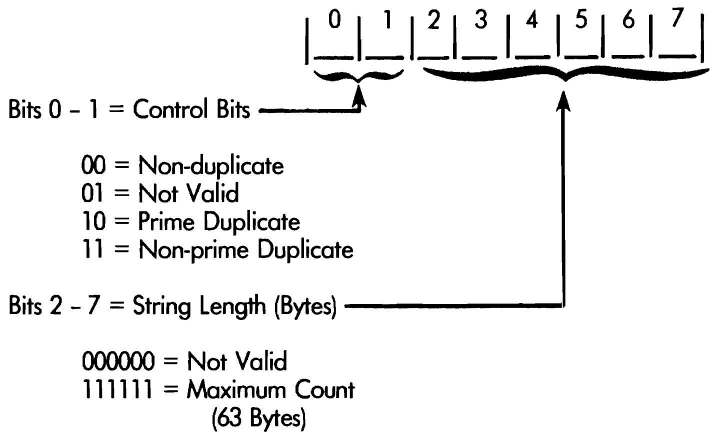


Figure 1-10
Control byte example

Note: Figures 1a and 1b have been adapted from examples in IBM's Technical Bulletin G360-1009.

Diskette and Tape Capacities

by John A. Gioannetti

Whether you use diskettes, magnetic tapes, tape cartridges, or a combination of these media, deciding how much off-line storage space you require can be difficult. For example, if you use diskettes as your backup media, you may (justifiably) feel that it takes too much time to initialize and label two 1D (single-density) diskettes when you can use one 2D (double-density) diskette. As a result, you waste off-line storage capacity backing up everything to 2Ds without checking to determine whether a 1D supplies adequate capacity.

If you choose magnetic tape or tape cartridge as your backup medium, the frustrating problems encountered when backing up to diskette persist. You might run out of space in mid-backup if you use a 600-foot tape, but when a 600-foot tape is sufficient, using a 3,600-foot tape wastes storage space.

The chart in Figure 1-11 lists the capacities of various diskette and magnetic tape formats and provides storage equivalents in alternative media. The

"Total Bytes" column on the chart helps you decide whether diskette or tape is the best medium and, in the case of diskette, which initialization format to use.

Note that your usable space for tapes is slightly less than that stated on the chart because some of the space is reserved for a header label for each file (a file locator automatically created by the system). Use the amounts specified on the chart as a guideline to maximum values.

If you don't know how many bytes you need to copy to off-line storage, use the S/36 CATALOG command to display the VTOC. The VTOC lists the space allocated for a file in blocks as well as records and furnishes the record length. Multiply the actual number of records in the file by the record length to determine how many bytes are needed for off-line storage of a particular file.

Another tip: on the S/36, you can save additional space when backing up to diskette by using the COMPRESS parameter in the SAVE command to compress duplicate character strings.

Figure 1-11

Diskette and maganetic tape storage capacities

Media Type	Format Type	Bytes Per Sector	Number of Sectors	Total Blocks	Total Bytes	Diskette Storage Equivalent			
						1D Format 1	1D Format 2	2D Format 1	2D Format 2
Diskette 1	1	128	1,924	96.20	246,272	1.0	0.81	0.25	0.20
Diskette 1	2	512	592	118.40	303,104	1.23	1.0	0.31	0.25
Diskette 2	1	256	3,848	384.80	985,088	4.00	3.25	1.0	0.81
Diskette 2	2	1,024	1,184	473.60	1,212,416	4.92	4.00	1.23	1.0
Magnetic Tape	300ft.			2,250.00	5,760,000	23.39	19.00	5.85	4.75
Magnetic Tape	600ft.			4,500.00	11,520,000	46.78	38.01	11.69	9.50
Magnetic Tape	1200ft.			9,000.00	23,040,000	93.56	76.01	23.39	19.00
Magnetic Tape	2400ft.			18,000.00	46,080,000	187.11	152.03	46.78	38.01
Magnetic Tape	3600ft.			27,000.00	69,120,000	280.67	228.04	70.17	57.01
Tape Cartridge	450ft.			16,857.00	43,200,000	175.42	142.53	43.85	35.63
Tape Cartridge	550ft.			20,625.00	52,800,000	214.40	174.20	53.60	43.55

Fixed Disk:
1 Block = 10 Sectors = 2,560 Bytes

8809 Tape Drive (Reel to Reel):
1,600 Bytes per Inch = 19,200 Bytes per Foot

6157 Tape Drive (Cartridge):
8,000 Bytes per Inch = 96,000 Bytes per Foot

Communications

CHAPTER

2

2

Transferring Files and Library Members via FTS

by John Fruetel

Code on diskette:
Procedure FTSPRC
RPG program FTSPRG
Screen format member FTSPRGFM

One day my company bought a small distributing company in the Pacific Northwest. The company was doing all its paperwork by hand, so we needed to set it up with some kind of on-line order entry and accounts receivable system that would be tied to our S/36 in central California. Because the company was small, we couldn't justify a big expense. My mission — which I had no choice but to accept — was to design a system and have the small new company up and running in a couple of months. Soon after I accepted, I had the strange feeling that somehow I had committed myself to something I knew nothing about. Little did I know my salvation would be my discovery of IBM's File Transfer Subroutines (FTS).

Before I discovered FTS, however, I explored other ways to bring the company on-line in a short time for a reasonable amount of money. As I saw it, there were only two options. The first was to set up the people in Washington with a remote workstation controller, terminals, printers, and a full-time leased line from here to there. The second was to give the people at the distributing company a small S/36 and to develop a departmental processing system. Because at that time I knew nothing about getting two S/36s to communicate with each other, I decided it would be easier to give the people at the remote site a 5294 controller.

However, when the phone company said that a dedicated line from central California to the Puget Sound area of Washington would cost more than $1,000 per month, I realized this approach would be much too expensive in the long run. It became apparent that the only cost-effective option would be to install a small S/36 at the remote site and to transfer data between it and our big S/36 here in California.

The first data transfer solution I investigated was IBM's Distributed Data Management (DDM). DDM lets a S/36 use another computer's files (either a S/36, a S/38, an AS/400, or a S/370) as if they were present on the local system. DDM was popular in the trade journals and seemed ideal for my application. Because a permanent leased line to the remote site was too costly, we could use DDM to upload and download files on the remote computer in batch processing once or twice a week. At first DDM's one-time charge of $2,000 seemed reasonable, but then the hidden costs of DDM began to multiply. DDM works best with a leased line, the system overhead for running DDM is quite high, and programmer and programming time must be allocated to maintain necessary network information.

I was discouraged until I came across FTS in Chapter 12 of *Using System/36 Communications* (SC21-9082). I had never heard of FTS before, but according to the manual, FTS would "allow a user application program to send or retrieve entire data files and library members from one System/36 to another." This sounded exactly like what I needed. And best of all, FTS is free! FTS is included on the Base Communications disk (feature 6001 for the 5360 and 5362, and feature 6047 for the 5363 and 5364), a part of SSP.

More and more companies are connecting S/36s and need to exchange files and library members. Until FTS, they had to use DDM to copy files, or they had to write their own RPG applications to transfer files and library members using bisynchronous communications. These RPG applications had two major problems: first, the hundreds of programs written by hundreds of different programmers were incompatible, and second, RPG could not access files and libraries directly, resulting in complicated programs and procedures. FTS has none of these problems. By definition, it is compatible among all S/36s. And because it is written in assembler, it handles files and libraries with ease. Furthermore, FTS assembler subroutines perform the transfers in less time than RPG takes.

With a sense of all of FTS's benefits, I was amazed that no one seemed to know very much about it. Even the IBM people with whom I spoke didn't seem to know that FTS existed. For some reason I don't completely understand, FTS has not been very popular even though it has notable capabilities — including a few, such as transferring library members, that DDM currently doesn't have on the S/36. To boost FTS' popularity, after describing FTS's functions and how to install it, I will explain how to use it in RPG programs and give an example.

What FTS Is and Does

The theory of operation behind FTS is remarkably simple and straightforward. The File Transfer Subroutines are IBM-supplied subroutines that can be incorporated into RPG II, COBOL, or assembly language programs to send or retrieve entire files or library members between two S/36s (Figure 2-1). An important characteristic of FTS is that a user-written program runs on only one system to perform the file or library member transfer. FTS automatically evokes a special FTS job on the target system to complete the transfer. Other transfer methods, including DDM and bisynch, require user-written programs running simultaneously on both systems. System A and System B must have a communications link established between them in the form of an Interactive Communication Feature (ICF) session (using either dial-up or leased point-to-point lines). You do not need to purchase ICF support from IBM, however, as everything you need is included free with the Base Communications Feature discussed earlier. FTS works with APPC (Advanced Program-to-Program Communications), BSCEL (Bisynchronous Communications Equivalence Link), Peer, and

Asynchronous Subsystems, but FTS does not enable an ICF session with the remote computer automatically. Also, FTS (Release 5.0 and later) optionally functions with APPN (Advanced Peer-to-Peer Networking) to transfer files between S/36s that are not necessarily adjacent nodes in the communications network (hereafter, I will use the word "file" to refer to both data files and library members).

Two different subroutines exist for FTS: SUBRF1 is used in COBOL or assembler programs, and SUBRF2 is used in RPG II programs. Because RPG is the predominant language in the S/36 world, my example is in RPG II using SUBRF2, but the concepts regarding the proper use of FTS apply to COBOL and assembler as well.

Installing FTS on a S/36

Installing FTS is easy. If you have Base Communications on your S/36, FTS is installed as well! If you have not installed Base Communications, you must do so for FTS to function. Installation is accomplished via Screen 21.1 of the CNFIGSSP procedure. The installation requires SSP diskettes, and you should apply the most recent PTFs after installing Base Communications support (see "FTS and PTF 05298," page 24). Installing Base Communications puts SUBRF1 and SUBRF2 in #LIBRARY.

FTS (Release 5.0 and later) puts a few load members and a single procedure in #LIBRARY. One of the load members, #FT#M1, is a message member that contains the text for the various FTS error codes. FTS is a little different from most IBM products that display error messages either at the system console or at the user's workstation. With FTS, if an error occurs (e.g., a user program requests a nonexistent file from a remote system), ERRMIC (the twelfth RLABL parameter in Figure 2-2) returns a Message Identifying Code (MIC) specifying the exact problem, and the user program must handle the FTS error accordingly. Your program can retrieve the text for FTS messages from #FT#M1 message member by using the // MEMBER statement and the IBM-supplied message-retriever subroutine SUBR23 (described in the RPG reference manual). Unfortunately, the message codes returned by FTS are not included in any of the message manuals. The codes and their meanings are listed only in Chapter 12 of *Using System 36 Communications* and in #FT#M1 in #LIBRARY.

With the exception of the message member #FT#M1, the other load members and procedure included in #LIBRARY cannot be referenced by user programs; they are used internally by SUBRF1 and SUBRF2 only.

Using FTS in an RPG Program

To use the FTS assembler subroutine to transfer data from one S/36 to another, code an EXIT operation to execute SUBRF2 (line 1 in Figure 2-2). Follow the EXIT operation by coding 13 parameters — each specified with the RPG II RLABL operation code — to specify which operations and functions FTS is to perform. Figure 2-3 shows the meaning of the general parameters in Figure 2-2 that are used whether you are transferring files or library members. Six other parameters — QUAL1 through QUAL6 — are used for both file and library copying, but the meaning of the parameters differs with the specific task. Certain parameter values are required for each type of transfer. Figures 2-4 and 2-5 show the parameter meanings when FTS is used to transfer files and library members respectively.

Another FTS parameter that requires further explanation is the PWORD parameter, which contains the password to use with your user ID when FTS attempts to log on to the remote system. The PWORD parameter is required only when the remote S/36 has password security active. FTS users must have a *user ID* that is the same on the remote system as on the local system; their *password* on the remote system, however, can differ from their local password. Unfortunately, password requirements can make FTS difficult to use when running a batch procedure with many users. FTS can validate everyone's user ID on the remote system, but there is no efficient way to enter and validate passwords. Hardcoding passwords causes difficulty. If more than one person is to use an FTS program in a batch procedure, passwords should be placed in a table or keyed in by the user as a parameter to the program.

Sample RPG Program with FTS

Procedure FTSPRC (Figure 2-6), in conjunction with the RPG program FTSPRG (Figure 2-7), sends and retrieves entire data files or library members from one S/36 to another.

Procedure FTSPRC runs in one of two ways, either interactively or in batch mode, depending on the parameters you specify on the prompt screen. If you specify the first three parameters, FTSPRC assumes FTS is to be run in batch mode and does not prompt you for more information (Figure 2-8a; Figure 2-8b contains the format member). If any of the first three parameters are not specified, FTSPRC assumes FTS is to run interactively and prompts the user for more information. Procedure FTSPRC then places the parameters regarding this file transfer into the LDA for program FTSPRG to use.

Program FTSPRG is simply a shell that calls the assembler subroutine SUBRF2, although it does contain logic to switch QUAL1 with QUAL4 and QUAL3 with QUAL6 if FTS is being used to receive a file from the remote system (QUAL1 and QUAL3 are source file/library names; QUAL4 and QUAL6 are target file/library names). The program does this because FTS normally refers to files as SOURCE and TARGET files. The SOURCE file is the file being sent or being received and is not necessarily a resident on the local system. I find this a little confusing, so I prefer to reference files as LOCAL or REMOTE. The logic in FTSPRG translates the SOURCE and TARGET names so the label LOCAL always refers to files on the local system, and REMOTE always refers to files on the remote system.

A Good Choice

As I said, FTS does not appear to be very popular in the S/36 world, and I don't understand why; the minor obstacles I've discussed here can be overcome easily. FTS is a powerful facility free for the asking for all S/36 owners with communications. It does not require complex programming, and it is an efficient means of communication. FTS even does a few things (e.g., transfer library members) that DDM cannot do on the S/36 at this time. If two S/36s need to communicate with each other from time to time in batch mode, FTS is an inexpensive substitute for DDM and an excellent choice for getting the job done. Because of FTS, that strange feeling I had when given my "mission" quickly turned into a satisfied feeling of accomplishment.

FTS and PTF 05298

Release 5.0 users should be aware of a potential problem with FTS if PTFs have been applied to their systems. PTF log number 05298 fixes a problem that FTS has with asynchronous communications support. But once PTF 05298 has been applied, that system can use FTS only with another system that also has PTF 05298 applied. In addition, Release 5.0 computers with PTF 05298 are unable to communicate with Release 4.0 or earlier S/36s.

If the computers you're planning to communicate with are not using SSP Release 5.0 with PTFs applied, and you are not planning to use asynchronous communications for FTS, you need to remove PTF 05298 from your system.

To do this, convince everyone to sign off the system, and make sure no batch jobs are running; removing a PTF requires a dedicated system. If you have service aid authority, key the following OCL statement:

```
PTF REMOVE,05298,ALLPTF
```

This OCL needs to be keyed every time a new PTF diskette is applied to the system (or until you install Release 5.1 or later).

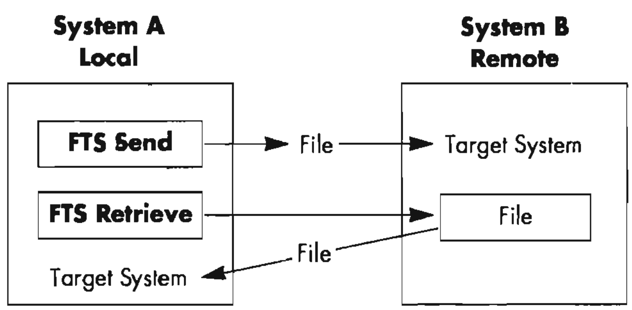


Figure 2-1 *Structure of FTS*

Figure 2-2

Calling sequence for SUBRF2 in RPG programs to transfer files

```
EXIT SUBRF2
RLABLFCODE1
RLABLQUAL18
RLABLQUAL26
RLABLQUAL38
RLABLQUAL48
RLABLQUAL56
RLABLQUAL68
RLABLREPL1
RLABLLOCNAM8
RLABLPWORD4
RLABLRCODE1
RLABLERRMIC8
```

Figure 2-3 *General parameters for FTS*

Parameter	Length	Required	Meaning
FCODE	1	Y	A one-character field that may contain an S or an R to indicate whether the user wants to send or retrieve a data file or library member.
REPL	1	N	A one-character field that specifies whether an existing file or library member should be replaced when the transfer is complete. Valid values for this field are Y, N, or blank. If the value isn't specifed (left blank), N is assumed.
LOCNAM	8	Y	This field specifies the logical name of the system with which you are communicating. The name given here must be the same as that specified with the CNFIGICF procedure. Also, this location must be currently active (i.e., it must have been ENABLED).
PWORD	4	?	This field should contain the password to use with your user ID when FTS attempts to log on to the remote system. This field is required only if the remote S/36 has password security active.
RCODE	1	Y	This one-character field contains the return code for FTS error meassages. Interpretations of the returned values are: 0: Normal completion, no problems 1: Problems at the local system 2: Problems at the remote system If 1 or 2 is returned, field ERRMIC indicates the error more specifically.
ERRMIC	8	Y	If RCODE returned 1 or 2, this field will contain an MIC code specifying the error.
APPN	1	N	This field specifies whether FTS should use (APPN) capabilities of locating a S/36 that may not be directly connected to the local S/36. Valid values for this field are Y (yes), N (no), or none. If no value is specified, the default is N.

Figure 2-4 *Parameter meanings for file transfer*

Parameter	Length	Required	Meaning
QUAL1	8	Y	This field contains the name of the file to be transmitted or received (source file name). File groups are not allowed here.
QUAL2	6	N	This optional field may contain the file date of the file name specifed in QUAL1.
QUAL3	8	Y	This field is required to be blank.
QUAL4	8	N	This optional field specifies the target file name for the transfer. If no value is specified, the source file name from QUAL1 is used.
QUAL5	5	N	Target system file date. If used, this field contains the file creation date for the file specified by QUAL4.
QUAL6	8	Y	This parameter is required to be blank for file transfer.

Figure 2-5 *Parameter meanings for library member transfer*

Parameter	Length	Required	Meaning
QUAL1	8	Y	Name of the library containing the member to be copied.
QUAL2	6	Y	Library member type to be transferred. Valid values are: SOURCE, PROC, LOAD, and SUBR.
QUAL3	8	Y	Name of the library member to be transferred.
QUAL4	8	N	Name of the library that will contain the transferred library member. If no value is specified for this field, the library name from QUAL1 is used.
QUAL5	6	Y	This field is required to be blank.
QUAL6	8	N	This field indicates the name of the library member at the target system. If this field isn't specified, the name from QUAL3 is used as the target system

Figure 2-6
Procedure FTSPRC

```
RLABLAPPN1
*
*                             Procedure FTSPRC
*
*  This procedure will transmit or receive entire data files or library
*  members to/from a remote S/36
*
*    Parameters
*
*              P1-S)end or R)eceive Data
*              P2-Enabled Location Name (MUST be enabled)
*              P3-Local File or Library Name
*              P4-Library Member Name (leave blank for files)
*              P5-Library Member Type (leave blank for files)
*              P6-Remote File or Library Name
```

```
*              P7=Remote Member Name (leave blank for files)
*              P8=Replace Existing File or Library Member (Y/N)
*              P9=Password (use if remote system has password security)
*             P10=Use APPN (Y/N)
*
// IFF '?1?'/ IFF '?2?'/ IFF '?3?'/ GOTO DOIT
*
// PROMPT MEMBER-FTSPRGFM,FORMAT-S1,START-1,LENGTH-'1,8,8,8,6,8,8,1,4,1'
// IF ?CD?=2003 RETURN
// ELSE IF ?CD?=2007 RETURN
// ELSE IF ?CD?=2005 EVOKE FTSPRC *ALL
// ELSE GOTO DOIT
// RETURN
*
// TAG DOIT
*
// LOCAL BLANK-*ALL
// LOCAL OFFSET-1,DATA-'?1?'
// LOCAL OFFSET-2,DATA-'?3?'
// LOCAL OFFSET-10,DATA-'?5?'
// LOCAL OFFSET-16,DATA-'?4?'
// LOCAL OFFSET-24,DATA-'?6?'
// LOCAL OFFSET-38,DATA-'?7?
// LOCAL OFFSET-46,DATA-'?8?'
// LOCAL OFFSET-47,DATA-'?2?'
// LOCAL OFFSET-55,DATA-'?9?'
// LOCAL OFFSET-68,DATA-'?10?'
*
// LOAD FTSPRG
// RUN
*
// IF ?L'59,1'?=0 RETURN
// MEMBER USER1-#FT#M1,LIBRARY-#LIBRARY
// * ?L'64,4'?
// PAUSE
*
```

Figure 2-7

Program FTSPRG

```
*        1         2         3         4         5         6         7       . 8
     H                                                                     FTSPRG
     H**********************************************************************
     H*                                                                    *
     H*                        Program FTSPRG                              *
     H*                                                                    *
     H*  This program will use SUBRF2 to either transmit or receive a file *
     H*  or a library member to/from another S/36  The remote S/36 must    *
     H*  have been enabled using the ENABLE procedure                      *
     H*                                                                    *
     H*  The parameters for SUBRF2 are defined in the LDA                  *
     H*                                                                    *
     H**********************************************************************
     H*
     I          UDS
     I                                        1   1 FCODE
     I                                        2   9 QUAL1
     I                                       10  15 QUAL2
     I                                       16  23 QUAL3
     I                                       24  31 QUAL4
     I                                       32  37 QUAL5
     I                                       38  45 QUAL6
     I                                       46  46 REPL
     I                                       47  54 LOCNAM
     I                                       55  58 PWORD
     I                                       59  59 RCODE
     I                                       60  67 ERRMIC
     I                                       68  68 APPN
     I*
     C**********************************************************************
     C*                                                                    *
     C*  In the 'C' specs of this program, I check to see if this program  *
```

```
C*  is going to receive a file/library member from the remote system    *
C*  If so  the program swaps QUAL1 with QUAL4 and QUAL2 with QUAL6      *
C*  Why?  Because FTS uses SOURCE and TARGET names and I prefer to use  *
C*  names as they appear on the local and remote system                 *
C*                                                                      *
C***********************************************************************
C*
C           FCODE     IFEQ 'R'
C                     MOVE QUAL1     TMP      8
C                     MOVE QUAL4     QUAL1
C                     MOVE TMP       QUAL4
C                     MOVE QUAL3     TMP
C                     MOVE QUAL6     QUAL3
C                     MOVE TMP       QUAL6
C                     END
C*
C                     EXIT SUBRF2
C                     RLABL          FCODE
C                     RLABL          QUAL1
C                     RLABL          QUAL2
C                     RLABL          QUAL3
C                     RLABL          QUAL4
C                     RLABL          QUAL5
C                     RLABL          QUAL6
C                     RLABL          REPL
C                     RLABL          LOCNAM
C                     RLABL          PWORD
C                     RLABL          RCODE
C                     RLABL          ERRMIC
C                     RLABL          APPN
C*
C                     SETON                     LR
C*
```

Figure 2-8a

Prompt screen for specifying FTS parameters

```
 ** Send or Receive Files or Library Members to/from a Remote System **

 Send or Receive Data (S or R)                                    _

 Remote Location Name (must be enabled)                           ________

 Local Library or File Name                                       ________

 Library Member Name (for transferring library members)           ________

 Library Member Type (for transferring library members)           ______

 Remote Library or File Name                                      ________   *

 Remote Member Name (for transferring library members)            ________   *

 Replace existing File or Library Member (Y/N)                    _          *

 Password                                                         ____       *

 Use APPN capabilities (Y/N)                                      _          *

                Cmd3-Return                Cmd5-Evoke
```

Figure 2-8b

SFGR specifications for FTS parameter prompt screen

```
*        1         2         3         4         5         6         7         8
SS1                     YY         Y                                 23CEG
D              70 1 7Y                 Y                        C** Send or Receive FileX
Ds or Library Members to/from a Remote System **
D              57 4 7Y                                          CSend or Receive Data (SX
D or R)
DFCODE          1 466Y  Y                     Y
D              57 6 7Y                                          CRemote Location Name (mX
Dust be enabled)
DLOCNAM         8 666Y  Y                     Y
```

```
D          57 8 7Y                                            CLocal Library or File NX
Dame   .
DQUAL1      8 866Y  Y                       Y
D          5710 7Y                                            CLibrary Member Name (foX
Dr transferring library members)
DQUAL3      81066Y  Y                       Y
D          5712 7Y                                            CLibrary Member Type (foX
Dr transferring library members)
DQUAL2      61266Y  Y                       Y
D          5714 7Y                                            CRemote Library or File X
DName                  .
DQUAL4      81466Y  Y                       Y
D           11476Y                                            C*
D          5716 7Y                                            CRemote Member Name (forX
D transferring library members)
DQUAL6      81666Y  Y                       Y
D           11676Y                                            C*
D          5718 7Y                                            CReplace existing File oX
Dr Library Member (Y/N)
DREPL       11866Y  Y                       Y
D           11876Y                                            C*
D          5720 7Y                                            CPassword              X
D .              .
DPWORD      42066Y  Y                       Y
D           12076Y                                            C*
DFA0004    5722 7Y                                            CUse APPN capabilities (X
DY/N)
DAPPN       12266Y  Y                       Y
DFA0003     12276Y                                            C*
D          112420Y                                            CCmd3-Return
D          102445Y                                            CCmd5-Evoke
```

Transmitting S/36 Object Code

by Gary T. Kratzer

program by Mel Beckman

Code on diskette:
Procedure MAKE$F
RPG programs MAKE$F, MAKMEM
Screen format member MAKE$FFM
Assembler subroutine SUBRCS

Utility MAKE$F eliminates inaccurate interpretation of transmitted code by using the IBM-supplied $FEFIX library member patch utility.

With the increasing popularity of public electronic mail and bulletin board systems, more and more people want to transmit S/36 object programs to other users via an electronic medium. Transmitting object programs rather than source programs lets recipients run the programs without having a compiler for the original source language.

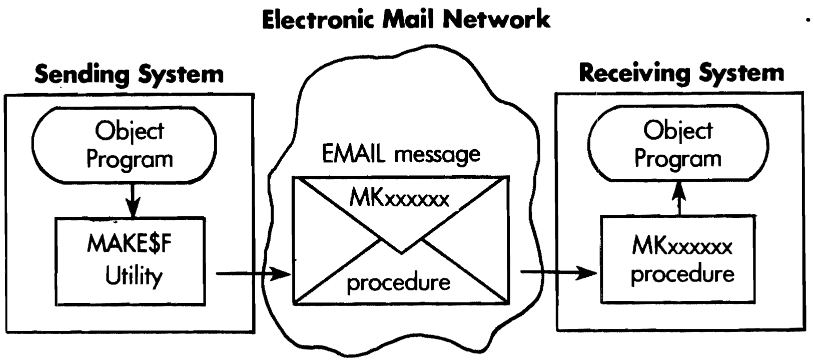
To transmit object code electronically, however, you must first overcome some difficulties. These difficulties involve representing data in a manner that ensures that the message goes through clearly. Object programs are stored as binary data containing nonprintable characters, while electronic mail is stored as text and is restricted to printable characters – the letters A to Z, numbers, and symbols. When you transmit object programs in binary form over communications lines, communications software misinterprets the nonprintable characters as control characters, garbling the data at the receiving end. Also, most electronic networks use ASCII character encoding, while the S/36 uses IBM's EBCDIC character set. Any message exchanged between a S/36 and an electronic network (or between two S/36s via an electronic network) usually undergoes translation from EBCDIC to ASCII and vice versa, a difficult proposition under the best of circumstances.

To solve these potential transmission problems, utility MAKE$F converts the binary data into hexadecimal "nibbles" and thereby uses two characters to represent each eight-bit binary byte. Under this scheme, the binary value 10011010 (X'9A') is transmitted as two characters, 9 and A. Any one-byte binary value can be represented hexadecimally by a combination of digits 0 through 9 and letters A through F. Because these are printable characters and because these characters survive EBCDIC/ASCII translation, they can be safely transmitted electronically as a plain text message. On the receiving S/36, the hex representation of the program is converted back into binary form and stored in a S/36 library member.

There are two types of S/36 object modules: O- and R-modules. The MAKE$F utility works only with R-modules, which are usually compiled or assembled — but unlinked — programs. If you want to use MAKE$F on a compiled or assembled program, specify NOLINK at compile time (which creates an R-module), use MAKE$F on the R-module, and after sending and running the MK*xxxxxx* procedure on the target machine, link the R-module using IBM's OLINK procedure to create an executable O-module.

Utility MAKE$F, which transforms the code for transmission and then restores it on the receiving end,comprise two sections. The first section consists of procedure MAKE$F, a prompt screen, program MAKE$F, and procedure MKSUBRCS. The first section is run on the sending system to create a patch procedure, MK*xxxxxx* (*xxxxxx* being the object program name), that will be transmitted to the receiving system. Utility MAKE$F's second section, run on the receiving system, consists of the transmitted patch procedure and program MAKMEM. The transmitted patch procedure contains the hex representation of the object program. When the patch procedure is run on the receiving system, it calls MAKMEM and $FEFIX to re-create the object program in a specified library. Figure 2-9 shows how the two sections of MAKE$F are interrelated.

Figure 2-9
Utility MAKE$F overview



A more detailed review of the first section of utility MAKE$F shows it to be straightforward. When you type in MAKE$F, prompt screen MAKE$F (Figure 2-10a; Figure 2-10b shows the prompt screen specifications.) On this screen, you enter the name of the program to be converted to hex nibbles and the library it resides in. You also may designate the library that will contain the object program on the receiving system. (The default library is #LIBRARY.) Procedure MAKE$F (Figure 2-11) then uses the IBM-supplied utility $MAINT to copy the object program in binary form into file BINARY. This file consists of records eight bytes long; the first seven records contain the *library directory entry*; the remaining records contain the binary object code. Program MAKE$F (Figure 2-12) converts all of these records into hex form and includes them in the MK*xxxxxx* patch procedure being created in file OUTPUT. Procedure MAKE$F then calls $MAINT to copy the procedure contained in file OUTPUT to the library containing the original object program. As a safety feature, program MAKE$F automatically generates special checksums that will be used by $FEFIX on the receiving system to detect any transmission-induced errors in the hex data. MAKE$F uses assembler subroutine SUBRCS to compute checksums.

Figure 2-13 shows a sample patch procedure, MKSUBR$C, that was produced on the sending system by the first section of utility MAKE$F. Procedure MKSUBR$C contains the hex representation for an assembler subroutine named SUBR$C.

At this point, MAKE$F has created a patch procedure ready for transmission on an electronic mail system or bulletin board system as a plain text message. The recipient of this message need only extract and run procedure MKSUBR$C to re-create the object program R-module SUBR$C in #RPGLIB (the library specified on the prompt screen).

Before you run procedure MKSUBR$C on the receiving system, you must have previously compiled program MAKMEM (Figure 2-14). Procedure MKSUBR$C first stores the hex representation of the library directory entry in the LDA, along with the number of records in the original BINARY file. Program MAKMEM can then re-create the BINARY file and insert the library directory entry into the first seven records. To do this, it retrieves the hex representation of the directory entry from the LDA and converts it into binary representation. The remaining records are written as binary zeros to reserve space for the program object code that will be inserted into R-module SUBR$C.

$MAINT runs next, reading file BINARY and creating a new library member in the target library (#RPGLIB in this example). $MAINT uses the first seven records from BINARY to create a library directory entry identical to the original library directory entry from the sending system, with the remaining records holding space in the currently empty object library member. Finally, patch utility $FEFIX inserts the actual binary object code into the newly created library member. When it finishes, object program SUBR$C exists in #RPGLIB, identical in every way to the original object program from the sending system.

Utility MAKE$F opens up electronic mail networks for exchanging object programs between S/36s. Because programs can be exchanged without being limited by the compilers available on the receiving end, useful routines written in uncommon languages like FORTRAN and assembler can be shared more easily.

Figure 2-10a
MAKE$F screen

```
 B                             MAKE$F PROCEDURE                       Optional-*

                    Creates a $FEFIX procedure to recreate the
                       object code for a subroutine member


  Name of member to be recreated                                    ________

  Name of library containing R member to be recreated               ________

  Name of library to contain the recreated member                   ________  *
    (this is also the library that will contain the MKxxxxxx proc)











 Cmd4-Put on job queue                              Cmd7-Cancel procedure
```

Figure 2-10b
Screen format member MAKE$FFM

```
*        1         2         3         4         5         6         7         8
     SMAKE$FP1           YY                                    DG
     DFL0001    16 130Y                               CMAKE$F PROCEDURE
     DFL0002    10 169Y                               COptional-*
     DFL0003    42 319Y                               CCreates a $FEFIX procedX
     Dure to recreate the
     DFL0004    35 422Y                               Cobject code for a subroX
     Dutine member
     DFL0005    63 7 3Y                               CName of member to be reX
     Dcreated
     DFL0006     6 768Y  Y                      Y
     DFL0007     2 775Y  Y          Y     Y
     DFL0011    63 9 3Y                               CName of library containX
     Ding R member to be recreated
     DFL0012     8 968Y  Y                      Y
     DFA0001    6412 3Y                               C  (this is also the libX
     Drary that will contain the MKxxxxxx proc)
     DFA0001    6311 3Y                               CName of library to contX
     Dain the recreated member
     DFA0002     81168Y  Y                      Y
     DFA0003     11178Y                               C*
     DFA0004    2124 2Y                               CCmd4-Put on job queue
     DFA0005    212453Y                               CCmd7-Cancel procedure
```

Figure 2-11

Procedure MAKE$F

```
*
*
*
*  MAKE$F is a utility that generates a $FEFIX procedure to re-create a given
*  subroutine or load member   $FEFIX is the IBM library patch utility that
*  exists on every S/36   When object members (R or O) are converted
*  to this format, each hex byte is represented by two characters   This
*  hex nibble representation will survive conversion between EBCDIC and
*  ASCII, and is thus a useful way to exchange object members on electronic
*  bulletin board services   MAKE$F computes checksums that are verified
*  when the object member is re-created, thus ensuring integrity of
*  transported object code
*
*
* Parm 1  name of module to be re-created
*      2  name of library containing input module (and to contain MK    proc)
*      3  name of library to receive re-created module when the MK..  proc
*         is executed (defaults to #LIBRARY)
*
// IF ?1?/ IF JOBQ-NO IF EVOKED-NO +
   PROMPT MEMBER-MAKE$FFM,FORMAT-MAKE$FP1  ?3'#LIBRARY'?
// IF ?CD?/2007 RETURN
// IF ?CD?/2004 JOBQ ?CLIB?,MAKE$F,?1?,?2?,?3?
// IF ?CD?/2004 RETURN
// IF JOBQ-NO IF EVOKED-NO +
   * '*MAKE$F* - Make a $FEFIX proc to recreate a subroutine member'
*
* Create a disk file containing the member to be cloned
*
// IF DATAF1-BINARY?WS? DELETE BINARY?WS?,F1
// LOAD $MAINT
// FILE NAME-BINARY?WS?,BLOCKS-25,EXTEND-25
// RUN
// COPY FROM-?2?,TO-DISK,FILE-BINARY?WS?,LIBRARY-R,NAME-?1?
// END
*
* Set up LDA and run MAKE$F to create $FEFIX procedure with checksums
*
// IF DATAF1-MAKE$F?WS? DELETE MAKE$F?WS?,F1
// LOCAL OFFSET-256,DATA-'?1?',BLANK-6          Module name
// LOCAL OFFSET-263,DATA-'?2?',BLANK-8          Library name
// LOCAL OFFSET-271,DATA-'?3'#LIBRARY'?',BLANK-8  Target library name
// LOCAL OFFSET-279,DATA-'?F'A,BINARY?WS?'?'      # of records in BINARY file
*
// LOAD MAKE$F
// FILE NAME-BINARY,LABEL-BINARY?WS?,RETAIN-S
// FILE NAME-OUTPUT,LABEL-MAKE$F?WS?,RECORDS-500,EXTEND-500
// RUN
*
* Place the MKxxxxxx $FEFIX procedure in the library
*
// LOAD $MAINT
// FILE NAME-MAKE$F?WS?,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-MAKE$F?WS?,TO-?2?,RETAIN-R
// END
```

Figure 2-12

Program MAKE$F

```
*       1         2         3         4         5         6         7         8
0001 H     064                      B       1                                MAKE$F
0002 F*
0003 F* Make $FFFIX procedure to recreate an R or O module
0004 F*                    .
0005 FBINARY  ID  F8000   8           DISK
0006 FOUTPUT  O   F9600  96           DISK
0007 E                   BIN8        4  8              8-byte binary chunks
0008 E                   BIN        32  1              Binary string
0009 E                   HEX        32  2              Hex string
0010 E                   HDI        16  1              Hex digit table 0-F
0011 E                   HNY        16  1              Hex nybble table 0-F
0012 E                   OCL     1  20 80              OCL text
0013 E                   OUT        96  1              Output work area
```

```
I*
I* $MAINT binary input file contains library member to be converted
I*
IBINARY
I                                        1   8 BIN8.X
I*
I* Redefine BIN8 and BIN
I*
I            DS
I                                        1  32 BIN8
I                                        1  32 BIN
I*
I* Redefine OUT array as a 97-byte field - last byte required by SUBRCS
I*
I            DS
I                                        1  96 OUT
I                                        1  97 OUTPUT
I*
I* Breakdown of hex address for incrementing
I*
I            DS
I                                        1   4 ADDR
I                                        1   1 ADDR1
I                                        2   2 ADDR2
I                                        3   3 ADDR3
I                                        4   4 ADDR4
I*
I* Local data area contains procedure parameters and size of file BINARY
I*
I           UDS
I                                      256 261 INPMEM
I                                      263 270 INPLIB
I                                      271 278 TRGLIB
I                                      279 2860#RECS
C/EJECT
C*
C* Initialization
C*
C                     MOVE '0000'    ADDR             Set starting address
C                     BITOF'01234567'HEX00   1        Make X'00' constant
C*
C* Initialize hex conversion tables
C*
C                     BITOF'01234567'X00     1        Constant X'00'
C                     MOVE X00       HNY              Clear hex values     ARRAY WITH X'00'
C                     BITON'7'       HNY,2            X'01'
C                     BITON'6'       HNY,3            X'02'
C                     BITON'67'      HNY,4            X'03'
C                     BITON'5'       HNY,5            and on
C                     BITON'57'      HNY,6            and on
C                     BITON'56'      HNY,7            ad nauseum
C                     BITON'567'     HNY,8
C                     BITON'4        HNY,9
C                     BITON'47'      HNY,10
C                     BITON'46'      HNY,11
C                     BITON'467'     HNY,12
C                     BITON'45'      HNY,13
C                     BITON'457'     HNY,14
C                     BITON'456'     HNY,15
C                     BITON'4567'    HNY,16
C                     MOVEA'01234567'HDI,1            Initialize hex digit
C                     MOVEA'89ABCDEF'HDI,9             table from 0-F
C*
C* Read directory entry (seven 8-byte records) and convert to hex
C*
C           1         DO   4         X       20       For the first 4 recs
C                     READ BINARY                      Read into BIN8 arry
C                     END                             End DO
C                     EXSR BINHEX                     Convert to hex
C                     MOVEAHEX,1     DIRA   64        Save as part A
C*
C                     MOVE HEX00     BIN              Clear the BIN array
C           1         DO   3         X       20       For the last 3 recs
C                     READ BINARY                      Read into BIN8 arry
C                     END                             End DO
C                     EXSR BINHEX                     Convert to hex
```

```
C                    MOVEAHEX,1     DIRB    48         Save as part B
C*
C* Output initial lines of procedure
C*
C                    EXCPTFRONT
C/EJECT
C*
C* Build and output HDR line   Format  HDR  cksm inpme00000
C*
C                    MOVEA*BLANKS   OUT,1              Clear output area
C                    MOVEA'HDR'     OUT,1              Built HDR line
C                    MOVEAINPMEM    OUT,11
C                    MOVEA'00000'   OUT,16
C                    EXIT SUBRCS                       Compute checksum
C                    RLABL          OUTPUT
C                    RLABL          CHKSUM
C                    MOVEACHKSUM    OUT,6              Insert checksum
C                    EXCPTOUTLIN                       Emit the line
C*
C* Build and output PTF line   Format  PTF cksm Tmodnam,lv,,libnam
C*
C                    MOVEA*BLANKS   OUT,1              Clear output area
C                    MOVEA'PTF'     OUT,1              Build PTF line
C                    MOVEA'R'       OUT,11
C                    MOVEAINPMEM    OUT,12
C                    Z-ADD12        X
C          ' '       LOKUPOUT,X                    11
C                    MOVEA',99,,'   OUT,X               (Rel lev always 99)
C                    ADD  5         X
C                    MOVEATRGLIB    OUT,X
C                    EXIT SUBRCS                       Compute checksum
C                    RLABL          OUTPUT
C                    RLABL          CHKSUM
C                    MOVEACHKSUM    OUT,6              Insert checksum
C                    EXCPTOUTLIN                       Emit the line
C*
C* Loop to produce DATA lines, with checksums  until done
C*
C          EOF       DOUEQ'Y'                          Do until EOF
C                    EXSR DATA                           Generate data line
C                    EXSR BUMP@                          Bump addr by X'20'
C                    END                                End
C*
C* Build and output END line with checksum   Format  END cksm
C*
C                    MOVEA*BLANKS   OUT,1              Clear output area
C                    MOVEA'END'     OUT,1              Build END line
C                    EXIT SUBRCS                       Compute checksum
C                    RLABL          OUTPUT
C                    RLABL          CHKSUM
C                    MOVEACHKSUM    OUT,6              Insert checksum
C                    EXCPTOUTLIN                       Emit the line
C*
C* End of program
C*
C          END       TAG
C                    SETON                      LR
C/EJECT
C*
C* Subroutine to build and output a DATA line
C* Format  DATA cksm addr hexdatastring
C*
C          DATA      BEGSR
C*
C                    MOVEAHEX00     BIN                 Clear binary string
C          1         DO   4         X       20          Get 32 bytes data
C  NLR               READ BINARY                    LR
C  NLR               END
C   LR               MOVE 'Y'       EOF      1          If LR, set EOF flag
C   LR     X         COMP 1                         11 If no records read
C   LR 11            GOTO DATAX                         Get out
C*
C                    EXSR BINHEX                        Convert to hex
C*
C                    MOVEA*BLANKS   OUT,1               Clear output area
C                    MOVEA'DATA'    OUT,1               'DATA'
```

```
C                     MOVEA'00'      OUT,11               'DATA      00'
C                     MOVEAADDR      OUT,14               'DATA      00 @@@@'
C                     MOVEAHEX,1     OUT,19               Copy hex datastream
C*
C                     EXIT SUBRCS                         Compute checksum
C                     RLABL          OUTPUT
C                     RLABL          CHKSUM  4
C                     MOVEACHKSUM    OUT,6                Insert checksum
C                     EXCPTOUTLIN                         Emit the line
C*
C           DATAX     ENDSR
C/EJECT
C*
C* Subroutine to convert binary data to hexadecimal
C*  Input  BIN    An arry of 32 binary bytes to be converted
C* Output. HEX    An array of 32 hex nybble pairs
C*
C           BINHEX    BEGSR
C*
C                     MOVEA*BLANKS   HEX                  Clear output area
C           1         DO   32        X                    For each bin byte
C*
C                     MOVE BIN,X     BITS    1             Get binary byte
C                     BITOF'0123'    BITS                  Clear high-order
C                     Z-ADD1         Y       20
C           BITS      LOKUPHNY,Y                      11   Lookup hex nibble
C                     MOVE HDI,Y     HEX,X                 Emit right hex nyb
C*
C                     MOVE BIN,X     BITS                  Get binary byte
C                     BITOF'4567'    BITS                  Clear low-order
C                     TESTB'0'       BITS             11   Shift
C   11                BITON'4'       BITS                   bits
C                     TESTB'1'       BITS             11     in
C   11                BITON'5'       BITS                     high
C                     TESTB'2'       BITS             11       nybble
C   11                BITON'6'       BITS                        to
C                     TESTB'3'       BITS             11          low
C   11                BITON'7'       BITS                          nybble
C                     BITOF'0123'    BITS                  Clear high-order
C                     Z-ADD1         Y
C           BITS      LOKUPHNY,Y                      11   Lookup hex nybble
C                     MOVELHDI,Y     HEX,X                 Emit left hex nyb
C*
C                     END                                 End DO
C                     ENDSR
C/EJECT
C*
C* Subroutine to bump a hex address by X'20'
C*  Input. ADDR  data structure containing four-digit hex address
C* Output: ADDR  is incremented by X'20'
C*
C           BUMP@     BEGSR
C                     Z-ADD1         X       20
C           ADDR3     LOKUPHDI,X                      10 Lookup hex digit
C                     ADD  2         X                    Bump for X'20'
C           X         COMP 16                    11       If overflow
C   11                SUB  16        X                    Then normalize
C                     MOVE HDI,X     ADDR3                And store
C*
C   11                Z-ADD1         X       20           If carry
C   11      ADDR2     LOKUPHDI,X                      10   Lookup hex digit
C   11                ADD  1         X                     Bump for X'100'
C   11      X         COMP 16                    12        If overflow
C   11 12             SUB  16        X                     Then normalize
C   11                MOVE HDI,X     ADDR2                 And store
C*
C   11 12             Z-ADD1         X       20            If carry
C   11 12   ADDR1     LOKUPHDI,X                      10    Lookup hex digit
C   11 12             ADD  1         X                      Bump for X'1000'
C   11 12             MOVE HDI,X     ADDR1                  And store
C*
C                     ENDSR
O/EJECT
O*
O* Front part of procedure
O*
```

```
0242 OOUTPUT  E               FRONT
0243 O                        OCL,1      80
0244 O                        INPMEM     31
0245 O        E               FRONT
0246 O                        OCL,2      80
0247 O                                   18 'R'
0248 O                        INPMEM     32
0249 O                        TRGLIB     52
0250 O        E               FRONT
0251 O                        OCL,3      80
0252 O        E               FRONT
0253 O                        OCL,4      80
0254 O                        #RECS      34
0255 O        E               FRONT
0256 O                        OCL,5      80
0257 O        E               FRONT
0258 O                        OCL,6      80
0259 O                        DIRA       65
0260 O                                   49 '99'
0261 O        E               FRONT
0262 O                        OCL,7      80
0263 O        E               FRONT
0264 O                        OCL,8      80
0265 O                        DIRB       49
0266 O        E               FRONT
0267 O                        OCL,9      80
0268 O        E               FRONT
0269 O,                       OCL,10     80
0270 O        E               FRONT
0271 O                        OCL,11     80
0272 O        E               FRONT
0273 O                        OCL,12     80
0274 O        E               FRONT
0275 O                        OCL,13     80
0276 O        E               FRONT
0277 O                        OCL,14     80
0278 O        E               FRONT
0279 O                        OCL,15     80
0280 O        E               FRONT
0281 O                        OCL,16     80
0282 O                        TRGLIB     50
0283 O        E               FRONT
0284 O                        OCL,17     80
0285 O        E               FRONT
0286 O                        OCL,18     80
0287 O                        INPMEM     22
0288 O        E               FRONT
0289 O                        OCL,19     80
0290 O        E               FRONT
0291 O                        OCL,20     80
0292 O*
0293 O* Variably built lines containing checksums
0294 O*
0295 O        E               OUTLIN
0296 O                        OUT        96
** Procedure text
// COPY LIBRARY-P,NAME-MKxxxxxx
// * 'Re-creating x-module xxxxxx in library xxxxxxxx'
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000000'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
// LOCAL OFFSET-273,DATA-+
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-targlib
// END
* Patch the new xxxxxx member to insert object code
// LOAD $FEFIX
// RUN
```

Figure 2-13

Sample patch procedure MKSUBR$C

```
// * 'Re-creating R-module SUBR$C in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'     Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'09E2E4C2D958C3404000000040000000000000600200000099000220000000001187'
// LOCAL OFFSET-273,DATA-+
'09171144310000000000000000000000000000086A0000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBR$C member to insert object code
// LOAD $FEFIX
// RUN
HDR  38AA SUBR$00000
PTF  CE87 RSUBR$C,99,,#RPGLIB
DATA 2A42 00 0000 E208E2E4C2D95BC300000000DD0000000000000000000000000000000000000000
DATA B306 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA E460 00 0040 E32D002D340800A9340100A1340200A5350100A91C0200CD023C0000CE1C0200
DATA E07A 00 0060 D1020F0100D100CF0E0100CF00D90E0100A90000202927232 11C18130F0B0703
DATA 46BB 00 0080 E32D005B000C350100A94D020200CDF2815C0D0100D100C0F184183C0000D21C
DATA B3A0 00 00A0 0200D502C0B700AA0C0000B600D30E0100D30000002D2927231E1A13110A0501
DATA FCDB 00 00C0 E32E00BA00D90D0100D300CFF7B2140C00008600CF0F0000B600D9370200D336
DATA B21F 00 00E0 0200CF0C00008700863501 00D16C0000000E01002824221E1A1614100E070501
DATA DCBB 00 0100 E33000BB000100D30F0100CF00D3F1876E0E0100A900DCC2010000C2020000C0
DATA A458 00 0120 870000350200 05BD4000F20114370200090F0000D30000302C22121009070301
DATA F4ED 00 0140 E30E00CA0009F10213E202013C0000D3F0870D0000000000000000000000000000
DATA 9E33 00 0160 00000000000000000000000000000000000000000000000000000000000000B01
DATA 6E9D 00 01B0 E306D0DC000000010000030000000000000DD0000000000000000000000000000
DATA 6F65 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 5622 00 01C0 C50000DC000000000000000000000000DD00D000000000000000000000000000000
DATA 76EB 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
END  2843
```

Figure 2-14

Program MAKMEM

```
*        1         2         3         4         5         6         7         8
0001 H     064                      B       1                             MAKMEM
0002 F*
0003 F* Create an empty $MAINT file with directory entry
0004 F* This file will be read by $MAINT to create an empty R- or O-module
0005 F* for patching by $FEFIX
0006 F*
0007 FBINARY  O   FB000   B           DISK
0008 E                    BIN8        7  8              8-byte binary chunks
0009 E                    BIN        56  1              Binary byte string
0010 E                    HEX        56  2              Hex nybble string
0011 I*
0012 I* Redefine BIN8 and BIN (which will contain the directory entry)
0013 I*
0014 I            OS
0015 I                                          1  56 BIN8
0016 I                                          1  56 BIN
0017 I*
0018 I* Local data area contains number of records to put in file BINARY
0019 I* and the library directory entry in hexadecimal notation
0020 I*
0021 I           UOS
0022 I                                        201 2D80#RECS
0023 I                                        209 320 HEX
0024 C/EJECT
0025 C*
0026 C* Initialization
0027 C*
0028 C                     BITOF'01234567 HEX00   1        Make X'00' constant
0029 C*
0030 C* Convert 56 directory entry bytes from hex nybbles to binary
0031 C*
0032 C                     DO   56        X       50        For each hexnyb pair
```

```
C                    MOVE HEX,X     NYB     1          Get right nybble
C                    EXSR CNVNYB                       Convert to binary
C                    MOVE BINNYB    BINARY  1          Save it
C                    MOVELHEX,X     NYB     1          Get left nybble
C                    EXSR CNVNYB                       Convert to binary
C                    TESTB'4'       BINNYB           11 Shift
C   11               BITON'0'       BINARY               bits
C                    TESTB'5'       BINNYB           11    in
C   11               BITON'1'       BINARY                 low nybble
C                    TESTB'6'       BINNYB           11      of BINNYB
C   11               BITON'2'       BINARY                   to high
C                    TESTB'7'       BINNYB           11        nybble of
C   11               BITON'3'       BINARY                       BINARY
C                    MOVE BINARY    BIN,X               Save binary byte
C                    END                               End DO
C*
C* Output directory entry to $MAINT binary file
C*
C                    DO   7         X      50          For each 8-bytes
C                    EXCPTBINREC                        Output a record
C                    SUB  1         #RECS               Decr total records
C                    END                               End DO
C*
C* Output zero-filled data records
C*
C                    DO   #RECS     X                  For the recs left
C                    EXCPTZEROS                         Output zeroed recs
C                    END                               End DO
C*
C* End of program
C*
C                    SETON                     LR
C/EJECT
C*
C* Subroutine to convert a hex nybble to a binary nybble
C*
C           CNVNYB    BEGSR
C                    MOVE NYB       BINNYB  1          Extract hex nybble
C                    BITOF'0123'    BINNYB              digits portion
C           NYB       IFLT '0'                          If not 0 through 9
C                    MOVE HEX00     BINNYB              Then clear bin nyb
C           NYB       COMP 'A'                       11 If 'A'
C   11               BITON'46'      BINNYB               then set X'A'
C           NYB       COMP 'B'                       11 If 'B'
C   11               BITON 467      BINNYB               then set X'B'
C           NYB       COMP 'C'                       11 Etc
C   11               BITON'45       BINNYB
C           NYB       COMP 'D'                       11
C   11               BITON'457      BINNYB
C           NYB       COMP 'E'                       11
C   11               BITON'456'     BINNYB
C           NYB       COMP 'F'                       11
C   11               BITON'4567'    BINNYB
C                    END
C                    ENDSR
O*
O* Binary directory records
O*
OBINARY   E                BINREC
O                          BIN8,X    8
O*
O* Binary-zero records
O*
O         E                ZEROS
O                          HEX00     1
O                          HEX00     2
O                          HEX00     3
O                          HEX00     4
O                          HEX00     5
O                          HEX00     6
O                          HEX00     7
O                          HEX00     8
```

Re-creating Subroutine SUBRCS

If you don't have assembler subroutine SUBRCS, you can re-create it with procedure MKSUBRCS (you don't need IBM's Assembler Language Program Product to install SUBRCS). You must have first compiled program MAKMEM (see Figure 2-14, page 38) to run MKSUBRCS. You need to run MKSUBRCS only once to create the SUBRCS subroutine.

```
// * Re-creating a module SUBRCS in library #RPGLIB
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-301 DATA-'00000071'     Number of #MAINT records
// LOCAL OFFSET-309 DATA-+
'[illegible]'
// LOCAL OFFSET-[illegible] DATA-+
'[illegible]'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-S,TO-#RPGLIB
// END
* Patch the new SUBRCS member to insert object code
// LOAD [illegible]
// RUN
HDR  [illegible] [illegible]
PTF  [illegible] [illegible],#RPGLIB
DATA [illegible] 00 0000 [illegible]
DATA [illegible] 00 0020 [illegible]
DATA [illegible] 00 0040 [illegible]
DATA [illegible] 00 0060 [illegible]
DATA [illegible] 00 0080 [illegible]
DATA [illegible] 00 00A0 [illegible]
DATA [illegible] 00 00C0 [illegible]
DATA [illegible] 00 00E0 [illegible]
DATA [illegible] 00 0100 [illegible]
DATA [illegible] 00 0120 [illegible]
DATA [illegible] 00 0140 [illegible]
DATA [illegible] 00 0160 [illegible]
DATA [illegible] 00 0180 [illegible]
DATA [illegible] 00 01A0 [illegible]
DATA [illegible] 00 01C0 [illegible]
DATA [illegible] 00 01E0 [illegible]
END  [illegible]
```

Transmitting 256-Byte Records with MSRJE

answered by Ed Giroux

Q I need help with multisession remote job entry (MSRJE) transmission from a S/36 to a mainframe. Specifically, I need a way for the S/36 to send records without breaking them into 80-byte records that must be rebuilt when received on the mainframe. Do you know how to get S/36 MSRJE to send records as long as 256 bytes?

A The length of records sent by S/36 MSRJE depends on the version of RJE, or JES (Job Entry Subsystem), used on the host end. JES2 supports records of up to 80 bytes; JES3, with some configuration effort, supports records of up to 256 bytes.

Some S/36 shops use a program that creates 80-byte transfer records by reading input files as variable-length data and using a compression routine, which lets one dataset contain multiple files and maximizes the amount of data transferred in one communications put. In fact, one of my clients includes a CRC-type (cyclic redundant check-type) counter at the end of the file so the decompression program on the receiving end can validate that the complete, correct file has been received.

Using Screen Formats in ICF Programs

answered by Mel Beckman

Q I have three S/36s hooked up in a multipoint environment, and I am attempting to use ICF (SNA Peer). After I enable the primary and one secondary location subsystem, I attempt to evoke a procedure on the remote system. I have written an RPG II program to pass the procedure name, library, and parameters using the ICF-defined screen format $$EVOK. However, I keep getting error SYS-5465, "Screen format used by program not found." I can't seem to find the error. Can you help me?

A ICF programs require that a continuation (K) line be coded for the workstation (Figure 2-15). If this line isn't coded, workstation data management attempts to find a screen format named $$EVOK, which doesn't exist. If the continuation line is coded, workstation data management treats the $$EVOK name as an ICF function.

Figure 2-15

Continuation line for an ICF workstation file

```
*       1         2         3         4         5         6         7    .    8
     FICFILE  CD          80              WORKSTN
     F                                                  KFMTS   *NONE
```

Suppressing Autodial Console Messages

answered by Nasser Shukayr

Q If you've done any autodial applications, you know that every time the ACU makes a call and the phone rings, SSP issues the message:

```
"SYS-8605 LINE-N CALL SUCCESSFUL TO"
```

If we made only three or four calls a day, this message would be no problem. But because we autodial all our branch offices repeatedly during the night, we must display and clear dozens of these SYS-8605 messages when we come in the next morning. Can you suggest a way to stop SSP from issuing this less than useful information?

A Enter the INFOMSG NO command at the system console at the end of the day to stop displaying informational messages. In the morning, enter INFOMSG YES to re-enable the display of informational messages.

Terminating BSC Jobs Automatically

answered by Jeff Silden

Q We have a problem with the Binary Synchronous Communications (BSC) on our S/36. When the system evokes the RPG II program that handles communications, the program sits in memory all day waiting for the phone to ring. Consequently, our 3:00 a.m. disk compress and library condense will not run, and no employees are here at the time to cancel the communications job.

We need to run the compress daily, but we cannot afford downtime during normal business hours. Is there a way to cancel this evoked job from within another procedure?

A Unfortunately, the BSC support IBM provides with the SSP is not sufficient for your purposes. BSC support will "hang" until one of three things happens: the line is disconnected, a call comes in, or you cancel the job (with the Attention key or an operator command).

An idea that comes to mind immediately would be to put a timer on the modem to (literally) turn out the lights at a predetermined hour. Although this technique would cause the program to go — almost immediately — to end of job, it also would generate an error message.

Perhaps your best solution would be to rewrite the application using the SSP-ICF Bisync Equivalence Link (BSCEL). This support is newer and more sophisticated, and therefore more costly. With BSCEL, you can add coding calls to the $$TIMER function. Such calls can allow the program to terminate automatically after a predetermined amount of time has elapsed without activity. The beautiful part about BSCEL (other than the fact that it solves your problem) is that you're just an ordinary batch BSC communications line and program from the other end's perspective — hence the name of the product.

VARYing Off Remote Devices on a Single Line

answered by Bob Tipton

Q I find it necessary to vary off and then vary on all remote workstations and printers on our S/36 several times during the day. I am tired of keying in 20 VARY commands, one for each remote device, every time I want to vary them off or on. Is there a better way to vary the status of remote devices?

A There is no need to vary the status of each remote device *individually* (i.e., V OFF,R1 or V OFF,P3). With one command, you can vary the status of remote control units on a communications line. When the status of a remote control unit is changed, the status of all devices attached to the control unit change. Thus, to change the status of remote control units (and therefore all devices attached to them) on line one, in console mode key the command

```
V OFF,,1
```

to vary off the control units or

```
V ON,,1
```

to vary on the control units.

Transmitting Orders from PCs to the S/36

answered by Matthew Henry

Q How can I, a wholesaler using a S/36, connect to clients who want to enter their orders electronically? Because my clients use PCs and midrange computers, I would have to be able to receive data files in both ASCII and EBCDIC formats. Ideally, I would like to send and receive a standard format via a store and forward mailbox system. I'm familiar with the mailbox options for the ASCII world; however, I don't know what kind of telecommunications are available for EBCDIC transmissions. If it's possible to connect directly to a S/36, must my clients also subscribe to the same network, or can a service such as DASnet connect all of us?

A There are three possible solutions to your problem. One, check out IBM's 9270 Voice Response Unit (VRU), a touch-tone phone entry system compatible with the S/36. The unit is user programmable and lets customers know things such as whether you stock an item they need. Two, you could use IBM's Interactive Communication Facility (ICF), which is standard equipment on the S/36. You'll also need a modem and access to DASnet. Using this method is akin to using an electronic answering service. Or three, you could connect your S/36 to your customers; use two numbers — one for PC dial-ins and one for midrange computer dial-ins. Connect the PCs dialing in to a PC that is locally attached to your S/36. The other number, used by those dialing in from a midrange computer, would be connected directly to your S/36, which would transfer data directly to a mailbox-type system on your S/36. The locally attached PC would need PC Support/36 to transfer the PC records to the S/36 to translate data from ASCII to EBCDIC automatically. You can automate the S/36 end easily, and, depending upon how frequently you need to exchange data, you can automate the PC end by running PC Support/36 on an hourly basis or daily basis.

Communicating with a PC Several Blocks Away

answered by Chuck Balsly and Ed Girou

Q We have a meat plant seven blocks down the street from our main office. The meat plant is on a leased line with a PC/AT running remote emulation. An attached Proprinter XL is configured as a 5256. Business conditions require a faster printer (in the 300-400 lpm range) at the plant, and I see my options as either continuing as we are with the remote devices or using the remote devices as local devices. If we run them as remote devices, we'll need a different controller at the meat plant because PC remote emulation won't support a high-speed twinaxial printer. Geographically, the devices are close enough to be local devices connected by twinaxial or fiber-optic cable. Do you have any suggestions for getting either type of cable laid for a reasonable amount of money?

A We have two possible solutions to your connectivity problem. Solution one is to obtain easement rights to bury a twinaxial cable. Be sure to use the proper lightning arresters, and expect a lengthy "settling down" time interval. For some reason, underground twinaxial cable is extremely sensitive and causes line drops. Also, insist that the cable be run as a single piece of wire (i.e., no cable splices). Solution two is to order an unloaded telephone line, which lets you use inexpensive 57.6 Kbps modems (i.e., direct-line drivers or short-haul modems, which cost approximately $750 on each end) with either a PC SDLC board, a 5X94 controller, or a used 5251 Model 12.

Communicating with PCs via the 5208 and DIAL/3X

answered by Chuck Balsly

Q Our problem involves a 5208 (ASCII link protocol converter) and DIAL/3X (Program 5799-PCE, Feature Code 9076, Release 1.0). We have connected our 5208 to a S/36 (model D24) and some PS/2s with DIAL/3X, which lets PS/2 users use an asynchronous modem to call the S/36. The PS/2s are configured in the 5208 as FILEXFR terminals, and we added new translation tables to accommodate special Danish characters. In general, this setup works fine and communications are established; however, there are times when the 5208 does not receive incoming calls properly, which means the PS/2 doesn't receive the sign-on display until we turn the device off and then on again. Have you ever heard of such a problem?

A Your hardware setup is okay. The IBM 5208 protocol converter is a private-label version of Telematics PCI-251 protocol converter; both units can use a variety of terminal emulation packages on remote PCs and PS/2s. DIAL/3X is used for two main reasons: for 5251 keyboard compatibility and for PC Support/36/38. If you don't need PC Support or file transfer capability, however, you may be better off choosing some other software because support for DIAL/3X is somewhat spotty. You can configure any commercial software emulator package, such as CrossTalk, to operate with the 5208, and you can resolve keyboard differences with any commercial key reassignment program. An alternative is to contact Telematics' local PCI distributor and purchase its software.

Transferring Files Between PCs and the S/36 via Asynchronous Communications

answered by Ed Girou

Q How can I call PCs in batches from a S/36 and transfer files up and down? Can I use the S/36 asynchronous communications support?

A The S/36 asynchronous communications support is extremely limited and isn't recommended for interactive or serious batch processing. Transferring a file from a PC to the S/36 using asynchronous communications doesn't use an error-correcting protocol (such as XMODEM), which results in undetected data errors caused by line noise. The best solution is to use a local PC to poll the remote PCs and transfer archival compressed files using an error-correcting transfer protocol (e.g., XMODEM, ZMODEM, DART, and FAST). Once the data files are on the PC, they can be decompressed and passed to the S/36 via PC Support/36 for further processing. This distributed approach also reduces the processing demands on your S/36.

The PC at your location could do the polling or just wait for the remote users to call it. If you decide to use the PC for polling, you'll need a communications program, such as DCA's CrossTalk Mk.4, that has a script language. If you decide to have remote users call the PC, use a BBS (bulletin board system) program, such as WildCat, which is inexpensive, easy to set up, and has solid security features.

Correcting DFU Zone Conversions When Using Display Station Passthrough

by Judy Miller

We are a target system for both a S/36 and a S/38 using Display Station Passthrough (DSPT) and APPC. Both remote systems were using a DFU program over one of our files that contained a 5.0 packed field, a 5.2 packed field, and a 1.0 packed field. The remote S/38 updated all fields correctly, but the S/36 always placed zero in the 1.0 field.

Finally, we found the solution in Appendix E of the *Programmer's/User's Workstation Guide.* The section on S/36 considerations with DSPT states that a problem exists with zone conversion from the S/36 local workstation controller. To solve the problem, simply re-create the DFU program with edit code 3 over all the numeric fields.

Adding an Inexpensive Asynchronous Modem to a 5363

by Don Bower

According to IBM, you need a serial adapter card (Feature Code 2620, $225) to use asynchronous communications on a S/36 5363. And, if you further follow IBM's advice, you need to attach an IBM 5853 1,200/2,400 bits per second (bps) asynchronous modem ($690) to that adapter card to complete your connection to the telephone network. Most people are unaware, however, that the 5363 essentially contains an embedded PC, with card slots identical to PC card slots. Thus, instead of using an IBM asynchronous adapter and external modem, I decided to plug a $200 Everex 1,200/2,400 bps internal modem board directly into my 5363 system unit. It works like a charm, and because the SSP support (Feature Code 6001) that lets you write programs to access the asynchronous port directly is free from IBM, I now have a simple, inexpensive, and elegant connection to the outside world at a $715 savings.

Internal modem boards are less expensive than external modem boards because they don't require a power supply, a switch, external indicators, or a case. And because extra external cables and boxes are eliminated, plugging the modular telephone line directly into the modem board in the back of your 5363 system unit makes the installation clean and uncluttered. With the capabilities of internal asynchronous modems improving to 4,800 and even 9,600 bps, you can expect to match low- and medium-speed synchronous modem performance at a fraction of the cost.

In addition, the 5363's PC chassis lets you use any PC-board device that looks like an asynchronous port to the PC. Fax boards (which cost less than $500) are one example of this kind of device.

Adding More Than 64 Remote Workstations

answered by Teresa Elms and Jeff Silden

Q I need more than 64 remote workstations on a S/36. Is there any way to add remote workstations via a local line and some "black boxes"?

A You can purchase protocol converters that attach to a twinaxial port and allow synchronous devices (such as 5250-compatible displays) or asynchronous devices (such as 3101 displays) to connect the S/36 via dial-up telephone lines. Then your only limitation would be the maximum number of local workstations. If your remote devices do not need to be operational all day, you might also consider using dial-up lines rather than dedicated lines to connect to your S/36 communications adapter; multiple remote devices could then share one remote workstation address.

Maximum Data Rates for S/36 Communications Adapters

answered by Mel Beckman, Jeff Silden, and Bob Tipton

Q We run a distributed data processing system that uses S/36s (5360s) as the major remote nodes. These machines are attached to our mainframe via a 3725 communications controller and leased lines running at 9,600 bps. One of our S/36s is a development machine and resides in-house. I am attempting to boost the line speed of this machine, which is attached via line driver to the 3725, from 9,600 bps to 19,200 bps.

The mainframe and the S/36 converse fine at 9,600 bps. But when I reconfigure the NCP program on the 3725 to support 19,200 bps and similarly boost speed on the line drivers, the S/36 will not respond to polling from the mainframe. We use external clocking, but I am sure there must be a speed setting for the communications lines on the S/36. I have come to the conclusion that it is probably a hardware switch on a card (probably the EIA line interface card) because I cannot find any software parameter to specify it. I have even taken the machine through IBM CE diagnostics.

I would like to know where and how to specify the S/36 communications line speed, even if it is just a pencil switch setting. I am continuously testing new communications methods and would like to test them at different speeds without an IBM engineer resetting the speed for me each time. Can you help me out?

A There are no internal speed settings for the communications lines on a S/36 using the EIA interface. Unless you have an IBM digital communications adapter, all S/36 communications with the eternal clock features rely on the modem (or modem eliminator) for line-speed clocking.

Your problem could be that you are attempting to exceed the maximum data rate for the communications equipment installed on your machine. The rules for determining communications data rates are complex. The following information summarizes the *S/36 Functions Reference Manual*, Chapter 12.

If your S/36 has an SLCA (Single-Line Communications Adapter) installed, the maximum data rate you can use is 9,600 bps for a 5360 system unit and 19,200 bps for a 5362 system unit. If you have an MLCA (Multi-Line Communications Adapter) installed on a 5360 system unit, line four

Data Conversion, Edits, and Validation

CHAPTER

3

3

Converting 24-Hour to 12-Hour Time, Part 1

by Charles Ackerman

Code on diskette:
RPG subroutine C24TO12A

With the idea that users should not have to feel like saying "tenhut" when trying to decipher the military time on a screen or on a report, I devised simple subprogram C24T012A (Figure 3-1) to convert the time from a 24-hour format to a 12-hour format. Because C24T012A is written in RPG II, it can be used by S/3X and AS/400 programmers.

S/36 programmers can use C24T012A as a subroutine in a program by eliminating the first two lines of code (PLIST and PARM) and the last line of code (SETON LR). The eight-byte work field TIMEAP will contain the time in HH:MM XX format.

C24T012A is a technique no shop should be without. With its simple way of converting the time to the more familiar 12-hour format, your report and screens can look more polished. You will eliminate the "technical" look the 24-hour or military format conveys.

Figure 3-1

Code to convert 24-hour to 12-hour time. (This code is contained in source member C24TO12A on diskette.)

```
*       1         2    .    3 .      . 4  .. ..  5 ..  .   6 ..     . 7   .  .. 8
     C              *ENTRY    PLIST
     C                        PARM           TIMEAP
      *
      *************************************************************
      *
      *
      *  -C24T012A-
      *
      *  Convert time of day into 12-hour format and put into alpha
      *
      *  field TIMEAP.
      *
      *
      *
      *************************************************************
      *
     C                        TIME           #TIME   60       TIME OF OAY
     C                        MOVEL#TIME     #HOUR   20       HOUR
     C              #HOUR     COMP 12                    90   SEE IF PM
     C    90        #HOUR     SUB  12        #HOUR            P.M.
     C                        MOVEL#TIME     TIME    40       > TIME OF OAY
     C                        MOVEL#HOUR     TIME             >FIELD.
     C                        MOVEL#TIME     #HOUR            SEE IF AM OR PM
     C              #HOUR     COMP 11                    90
     C    90        #HOUR     COMP 24                      90
     C                        MOVE 'AM'      AMPM    2        AM OR PM FIELO
     C    90                  MOVE 'PM'      AMPM
     C                        MOVELTIME      WRK3    3
     C                        MOVE ':'       WRK3
     C                        MOVE TIME      WRK5
     C                        MOVELWRK3      WRK5    5
     C              #HOUR     COMP 0                         90
     C    90                  MOVEL'12'      WRK5
     C                        MOVELWRK5      TIMEAP  8
     C                        MOVE AMPM      TIMEAP
     C                        MOVELTIMEAP    WRK1    1        LEADING ZERO?
     C              WRK1      COMP '0'                       95 YES .
     C    95                  MOVEL' '       TIMEAP            BLANK IT OUT.
     C                        SETON                      LR
```

Converting 24-Hour to 12-Hour Time, Part 2

by Jeff Cole

Code on diskette:
RPG subroutine C24TO12B

Converting 24 Hour to 12 Hour Time, Part 1 shows a subprogram to convert the time from a 24-hour format to a 12-hour format. S/36 programmers may find the C24T012B subroutine in Figure 3-2 easier and quicker to implement into existing software.

The C24TO12B subroutine moves the hour portion of the system time into PMTEST and compares the hour to 12. If the hour is greater than 12, 12 is subtracted from the hour. The time then prints with the appropriate 12-hour abbreviation.

Figure 3-2

Code to convert 24-hour to 12-hour time. (This code is contained in source member C24TO12B on diskette.)

```
•        1         2         3         4         5         6         7     .   8
     C*
     CSR         TIMESR    8EGSR
     CSR                   TIME           TMPTME  60
     CSR                   MOVELTMPTME    PMTEST  20
     CSR                   MOVELTMPTME    UTIME   40
     CSR         PMTEST    COMP 12                   12  11
     CSR 12                SU8  1200      UTIME
     CSR 11                SETON                     12
     CSR                   ENOSR
     .

     0                                    5 'TIME:'
     0                        N12UTIME   16 '0    &A.M.'
     0                         12UTIME   16 '0    &P.M '
```

Converting 24-Hour to 12-Hour Time, Part 3

by Carson Soule

Code on diskette:
RPG subroutine C24T012C

I do not offer here yet another time conversion routine. Instead, I offer a more structured version (program C24T012C in Figure 3-3) of a previously published technique because I believe structured code is much clearer and as a result more reliable and transportable. In my version, only one indicator (86) connects the calculations and the output and only two indicators are used in the time conversion.

Indicator 50 is set on when the system time is moved into WTIM. WTIM is truncated to the hours and minutes (WHR) and compared to 1200 (noon). If the hour is greater than or equal to 1200, indicator 51 is set on, PM is moved into WPM, and 1200 is subtracted from the time to arrive at the 12-hour format. If indicator 50 is on but 51 isn't, it must be before noon rather than after noon, so AM is moved to WPM. Then WHR is compared to 0100 to determine whether the time is between midnight and 1:00 AM. If this is the case, 1200 is added to the time. At this point, the time conversion is complete, indicator 86 is set on, and the headings are printed.

Figure 3-3

Code to convert 24- into 12-hour time. (This code appeares in source member C24T012C on diskette.)

```
*  .      1 .       2 .       3 .       4         5        . 6 ..     . 7       8
     C   TIME - GET SYSTEM TIME AND SIMULATE 1P OUTPUT
     C*
     CSR         TIME      BEGSR
     C           WFRS      COMP 0                          50IF FIRST TIME
     C   50                TIME           WTIM    60         THEN GET TIME
     C   50                MOVELWTIM      WHR     40         EXTRACT HR/MIN
     C   50      WHR       COMP 1200                 51  51IF HOUR->NOON
     C   50 51             MOVE 'PM'      WPM      2         THEN PM
     C   50 51             SUB  1200      WHR                THEN ADJUST HR
     C   50N51             MOVE 'AM'      WPM                ELSE AM
     C   50      WHR       COMP 0100                   51    IF < 1:00
     C   50 51             A00  1200      WHR                THEN AOJUST HR
     C   50                ADD  1         WFRS               NOT FIRST TIME
     C   50                SETON                     86      PRINT HEAOINGS
     C  N50                SETOF                     86      ELSE ENO HOR PR
     C                     ENDSR
     C*
     0        D  2     86
     0       OR        OF
     0                                     4 'TIME'
     0                           WHR      11 ' : '
     0                           WPM      14
     0*
```

Converting and Editing 24-Hour to 12-Hour Time in OCL

by Heather G. Quinn

Code on diskette:
Procedure C24T012

In the history of BitStop, you have published a number of time conversion techniques. Here is one more for the collection. On the S/36, converted time can be handled external to any program and passed into a program via the LDA. Because LDA data is available at program load time (before the first calculation cycle), this data may be used on 1P-conditioned O-specs in RPG II programs.

Procedure C24T012 in Figure 3-4 may be called from any other procedure. It takes the system time, converts it to 12-hour format, and inserts a colon(:) between the hours and minutes of the converted time. Upon return to the calling procedure, the edited data is available in positions 1 through 7 of the LDA and in returning parameter 3. Thus, the converted and edited time may be used as you will, in a procedure or a program.

Figure 3-4

Procedure C24T012

```
// IFT ?TIME?>115959 EVALUATE P2=PM
// IFT ?TIME?>125959 EVALUATE P1,6=?TIME?-120000
// ELSE IFF ?TIME?>005959 EVALUATE P1,6=?TIME?+120000
// LOCAL OFFSET-1,DATA-'?1'?TIME?'?'
// EVALUATE P3='?L'1,2'?:?L'3,2'??2'AM'?'
// LOCAL OFFSET-1,BLANK-7
// LOCAL OFFSET-1,DATA-'?3?'
// RETURN *ALL
**
** Procedure "C24T012"   Convert 24-hour System Time to 12-hour AM/PM time, and
**                      return to calling procedure with converted, edited time
**                      in positions 1-7 of the Local Data Area and in
**                      Parameter 3   (Use of this procedure is limited to the
**                      S/36 only, because of parameter manipulations  Allows
**                      converted time to be used in any manner in any proc
**                      or RPG II program, including use on 1P-conditioned
**                      Output specs.)
**
```

Validating Days in Dates in OCL

by Edward Schroeck

Code on diskette:
Procedure VALDAY

A previously published BitStop presented a S/36 procedure that validated the day portion of a date in the MMDDYY format. Procedure VALDAY (Figure 3-5) accomplishes the same thing and more. This procedure will not allow the month or the day to be zero, validates the month portion of the date, and accommodates February 29 as valid for leap years.

I establish an array for the days in the LDA and issue a prompt screen. If either the entered day or month is zero, switch 1 comes on, the procedure displays an appropriate error message, and the prompt screen is redisplayed for re-entry.

If the month is February, the procedure divides the year by 4 and then multiplies that result by 4. If the value so obtained is the same as the year the user entered, it's a leap year, so 29 is inserted in positions 3 and 4 of the array.

The procedure then validates the day and checks to see that the value entered for month is less than 12. If either of these tests fails, the procedure again loops for re-entry of the date.

Note that a century year must be a multiple of 400 (not just a multiple of 4) to be a leap year. However, procedure VALDAY recognizes any year that ends in 00 as a leap year. Because we are coming up on the year 2000 (which will be a leap year), this should present no problems. However, the procedure will produce unpredictable results if you use it to validate dates in the years 1700, 1800, and 1900.

Figure 3-5

Procedure VALDAY

```
* DATE VALIDATION USING ONLY OCL
// LOCAL BLANK-*ALL
// TAG START
// SWITCH 00000000
// LOCAL OFFSET-1,DATA-'312831303130313130313031'
```

```
// PROMPT MEMBER-TDATE.FORMAT-SCRN01,LENGTH-'2,2,2,13,13'
// IF ?1?=0   EVALUATE P4='MONTH = ZERO '
// IF ?1?=0  SWITCH 10000000
//  ELSE   EVALUATE P4='
// IF ?2?=0  EVALUATE P5='DAY = ZERO
// IF ?2?=0  SWITCH 10000000
//  ELSE  EVALUATE P5='
// IF SWITCH-1    GOTO START
*
*   IF FEBRUARY AND LEAP YEAR MOVE 29 TO DAYS IN FEBRUARY.
// IFF ?1?=02  GOTO START2
// EVALUATE P7=?3?/4
// EVALUATE P8=?7?*4
// IF ?3?=?8? LOCAL OFFSET-3,DATA-'29'
* END FEBRUARY LEAP YEAR CHECK
*
// TAG START2
// EVALUATE P6=?1?*2-1
// IF ?2?>?L'?6?,2'?   EVALUATE P5='INVALID DAY
// IF ?2?>?L'?6?,2'?   SWITCH 10000000
//  ELSE  EVALUATE P5='
// IF ?1?>12  EVALUATE P4='INVALID MONTH'
// IF ?1?>12  SWITCH 10000000
//  ELSE EVALUATE P4='
// IF SWITCH-1   GOTO START
```

Testing for Numeric Values, Part 1

by Gerry Karpen

Code on diskette:
RPG code NUMCK1

I have a routine to test a field for all numeric values. My method involves three calculation lines and a 15-element array into which the field to be tested is moved (Figure 3-6).

Because the values zero through 9 are an F0 to an F9 hexadecimal value in the computer, one needs only to test the literal characters zero and 9 against every element in the array to determine whether there is something greater than or less than those two digits. Using the LOKUP command, RPG can test every character in the array against the Factor 1 digit. If a number less than F0 is found, indicator 02 is set on. If indicator 02 is not on, I test for a value greater than F9. Indicator 02 is set on if a value greater than F9 is found. Based on the condition of indicator 02 after these two calculations have been performed, the field can be determined to be either numeric (indicator 02 is not on) or non-numeric (indicator 02 is on). Subsequent logic may then use this indicator to condition calculations. (Note: Because an alphanumeric field cannot be used in computations, the tested field should be defined twice in the I-specs — once as alphanumeric for test purposes and once as numeric — for computational purposes if the test proves it to be all numeric.)

Figure 3-6

Code to test for all numeric data. (This code appears in member NUMCK1 on diskette.)

```
*        1    .    2          3  . .   4          5          6          7          8
     C   01                  MOVEAFLD     AR
     C   01        '0'       LOKUPAR                     02
     C   01NO2     '9'       LOKUPAR                   02


     O                           FLD      15
     O                   01 02            39 'NOT NUMERIC      '
     O                   01NO2            39 '    NUMERIC      '
```

Testing for Numeric Values, Part 2

by H. C. Currie

Code on diskette:
RPG code NUMCK2

You can construct a simple routine to test for an all-numeric field if you take advantage of certain features of the RPG MOVE operation. Specifically, when an alphanumeric field is moved to a numeric field, only the digit portion of each alphanumeric character is moved to the digit portion of the corresponding numeric character. The zoned portion of the numeric character is set automatically to hex F. (The only exception to this is the rightmost numeric character, which is set to hex D if the zone of the rightmost alphanumeric character is hex D). Thus, if you perform such a move on a test field, the numeric digits will be unchanged by the move because all numbers (0 to 9) already contain the hex F zone; only alphanumeric digits will change.

The subroutine NUMCK2 (Figure 3-7) uses this fact to advantage in testing for non-numeric fields. In NUMCK2, the field to be tested (ALPHA1) is moved to a numeric field (NUMER1) of the same field size. During the move, any non-numeric digits will have the zoned portion of the digit changed to hex F (or possibly hex D, if it is the rightmost digit in NUMER1 and the zone portion of the corresponding digit in ALPHA1 is hex D). Field NUMER1 is moved to field ALPHA2 because the RPG II compiler does not permit comparisons between numeric and alphanumeric fields.

Then ALPHA1 is compared to ALPHA2. If the test field is numeric, ALPHA1 will be identical in content to ALPHA2, and indicator 66 will be set off. If the test field is non-numeric, ALPHA1 will be different from ALPHA2, and indicator 66 will be set on.

Subroutine NUMCK2 can accommodate all numeric field sizes, up to the maximum of 15 digits. And subroutine NUMCK2 will flag numeric fields that contain one or more embedded blanks as non-numeric.

Unlike similar routines, signed numeric fields will not be kicked out as non-numeric fields by NUMCK2. If you are editing fields that should contain only unsigned numeric data, you may want signed numbers to be flagged as non-numeric. You can accomplish this by using the MOVEL operation instead of the MOVE operation to move the test field to the ALPHA1 field. Using the MOVEL operation in this situation works as long as the field you are testing has a length less than 15. If the field length is equal to 15, the MOVEL is identical in effect to the MOVE because the receiving field (ALPHA1) has a field length of 15.

Figure 3-7

Code to test for all numeric data. (This code appears in member NUMCK2 on diskette.)

```
*        1           2          3          4          5          6          7 .     . 8
     C                          MOVE *ZERO     ALPHA1


     C                          MOVE ANYFLD    ALPHA1
     C                          EXSR NUMCHK


     CSR          NUMCHK        BEGSR
     CSR                        MOVE ALPHA1    NUMER1 150
     CSR                        MOVE NUMER1    ALPHA2 15
     CSR          ALPHA1        COMP ALPHA2              6666        .
     CSR                        MOVE *ZERO     ALPHA1 15
     CSR                        ENDSR
```

Converting Gregorian and Julian Dates and Validating Dates

by Chuck Lundgren

Code on diskette:

Program @DTE1, @DTE2, @DTLY

There is an easy way to validate dates in entry programs that involves converting the date from the Gregorian format (MMDDYY) to the Julian format (YYnnn, where nnn = chronological day number within year). First save the Gregorian date in a work field. Next, convert the Gregorian date to the Julian date using routine @DTE1 (Figure 3-8a); then convert the Julian date back to the Gregorian date using routine @DTE2 (Figure 3-8b). Afterward, compare the saved Gregorian date to the newly created Gregorian date (the date converted to and from the Julian date). If the old date and the new date are the same, the date is valid; if they differ, the date is invalid. Routine @DTLY (Figure 3-9) is used by both routines to determine if the year is a leap year.

Figure 3-8a

RPG subroutine @DTE1 that converts a Gregorian date to a Julian date

```
*        1   . . 2          3          4        . 5          6 . .    . 7          8
     C            @DTE1         BEGSR
     C            #MM           ADD  2         #TEMP1  50
     C                          MULT 3055      #TEMP1
     C                          DIV  100       #TEMP1
     C            #TEMP1        SUB  91        #DDD    30
     C                          MOVEL#CC       #CCYY   40
     C                          MOVE #YY       #CCYY
     C                          EXSR @DTLY
     C            #MM           IFGT 2
     C                          SUB  2         #DDD
     C                          ADD  #LY       #DDD
     C                          END
     C                          ADD  #DD       #DDD
```

```
     C                   Z-ADD#DDD      #YYDDD  50
     C                   MOVEL#YY       #YYDDD
     C                   ENDSR
```

Figure 3-8b

RPG subroutine @DTE2 that converts a Julian date to a Gregorian date

```
*    1         2         3         4         5         6         7         8
     C         @DTE2     BEGSR
     C                   MOVEL#YYDDD    #YY
     C                   MOVE #YYDDD    #DDD    30
     C                   MOVE #CC       #CCYY   40
     C                   MOVEL#YY       #CCYY
     C                   EXSR @DTLY
     C         #LY       ADD  59        #TEMP1
     C         #DDD      IFGT #TEMP1
     C         2         SUB  #LY       #TEMP2  50
     C                   ADD  #DDD      #TEMP2
     C                   ELSE
     C                   Z-ADD#DDD      #TEMP2
     C                   END
     C         #TEMP2    ADD  91        #TEMP3  50
     C         #TEMP3    MULT 100       #TEMP1
     C         #TEMP1    DIV  3055      #MM
     C         #MM       MULT 3055      #TEMP1
     C                   DIV  100       #TEMP1
     C         #TEMP3    SUB  #TEMP1    #DD
     C                   SUB  2         #MM
     C                   ENDSR
```

Figure 3-9

RPG subroutine @DTLY determines if the given year is a leap year

```
*    1         2         3         4         5         6         7         8
     C         @DTLY     BEGSR
     C                   Z-ADD0         #LY     10
     C         #CCYY     DIV  4         #TEMP0  40
     C                   MULT 4         #TEMP0
     C         #CCYY     COMP #TEMP0                    91
     C   91              Z-ADD1         #LY
     C         #CCYY     DIV  100       #TEMP0
     C                   MULT 100       #TEMP0
     C         #CCYY     COMP #TEMP0                    91
     C   91              Z-ADD0         #LY
     C         #CCYY     DIV  400       #TEMP0
     C                   MULT 400       #TEMP0
     C         #CCYY     COMP #TEMP0                    91
     C   91              Z-ADD1         #LY
     C                   ENDSR
```

Formatting Left-Hand Negative Signs

by Elliot Weinshenker

Code on diskette:
RPG code NEGLFT

"If IBM had intended for us to have a floating negative sign on the left, they'd have provided us with an edit code for that." Such was my argument to management, who still insisted on having a negative sign on the left for our month-to-month variance figures on several different reports.

Once I resigned myself to the effort, I found that writing the routine (Figure 3-10) to provide a floating negative sign on the left was easier than I expected. If the value in question (LICS) is less than zero, I multiply it by -1 to make it positive (indicator 30 retains the fact that it was negative) and then move it to an alpha array (A08). The routine loops to inspect each element of the array (beginning at the left) and replaces each leading zero with a blank space. When the first nonzero digit is encountered, the program backs off the subscript (X) one position and moves in the negative sign. The resulting array can be printed as an alpha field with zeroes suppressed and with the negative sign just to the left of the number.

Figure 3-10

Routine to place a floating negative sign on the left. (This code appears in source member NEGLFT on diskette.)

```
*     . 1       . 2  . . . . 3  .  . 4 . .    5  .     6 .      7        8
     C          LICS      COMP *ZEROS                30
     C*
     C                    Z-ADD1         X       20
     C*
     C    30              MULT -1        LICS
     C                    MOVE LICS      AMT8    8
     C                    MOVEAAMT8      A08
     C*
     C          LOOP01    TAG
     C*
     C          X         COMP 9                        80
     C          A08,1     COMP *ZEROS               89
     C    80 30           SETON                     70
     C    89
     COR  80              GOTO TAG01
     C*
     C          A08,X     COMP *ZERO                    40
     C    40              MOVEA' '       A08,X
     C    40              ADD  1         X
     C    40              GOTO LOOP01
     C    30              SUB  1         X
     C    30              MOVEA'-'       A08,X
     C          TAG01     TAG
     C                    MOVEAA08       S1      8
     C*
     C                    Z-ADD*ZEROS    LICS
     C                    MOVE *ZEROS    ATM8
     C*
```

Overriding RPG's Date Edit Code

answered by Bob Tipton

Q My company was recently purchased by a Fortune 500 company. As a result, our new corporate MIS department has sent down a reporting standard edict: "All dates on all reports are to be separated with dashes." Because the standard Y edit code (which we use on our S/36 reports) separates dates with slashes, not dashes, we are faced with having to set up edit words for every date on every report just to change the date separator characters. Is there an easy way to change the date separator on the S/36?

A There is a much easier way than setting up edit words to edit your report dates with dashes instead of slashes. The RPG compiler has the ability to override the date edit code on a program-by-program basis. To separate your dates with dashes instead of slashes, simply key a dash (–) in column 20 of the H-specs in every program that outputs a date with a Y edit code. Then recompile the programs, and your dates will be edited with dashes instead of slashes.

Converting Date Format from MMDDYY to YYMMDD in OCL

by Grace E. Sogomian

Converting date formats from MMDDYY to YYMMDD can be performed with a single EVALUATE statement in a S/36 procedure because when division is performed in an EVALUATE statement, the remainder is dropped. The EVALUATE statement used is:

```
// EVALUATE  P1,6=?DATE?0000+(DATE/100)
```

Although the SET command in OCL accomplishes the same task, our DP department prefers to use the EVALUATE statement because it does not alter the session date.

Formatting Dates

by Timothy J. Plas

Programming tricks, such as the famous one-line RPG trick to convert dates between YYMMDD and MMDDYY format, are notorious for the problems they can cause during program maintenance. But if you use this particular trick, you could be adversely affecting system performance as well.

This frequently published trick uses RPG's truncation properties and some "magic" constants: YYMMDD MULT 100.0001 MMDDYY or MMDDYY MULT 10000.01 YYMMDD. You define the date fields with six digits and zero decimal positions. Because this conversion trick is so compact, we used it in many AS/400 applications — and got 150,000 to 200,000 decimal data size exceptions every day in the Performance Tools Exception Occurrence Summary Report.

The technique's reliance on truncation is precisely what causes the problem. The compiler builds in an exception-handling routine that says "do nothing but truncate the result field." This exception routine invokes system overhead functions that adversely affect performance. Instead of relying on truncation for date conversion, you can use the four lines of MOVE and MOVEL logic in Figure 3-11 to reformat a date — which is also executed many times faster than the truncation trick.

Figure 3-11

Code to reformat date

```
*... .. 1 .  . 2        . 3 .  . 4 . . . 5 ...  . 6 .. .. 7 .... 8
     C                      MOVEL MMDDYY  WORK4    40
     C                      MOVE  MMDDYY  WORK2    20
     C                      MOVEL WORK2   YYMMDD   60
     C                      MOVE  WORK4   YYMMDD
```

Computing Day of the Week in OCL

by Mark Allen

Code on diskette:
Procedure CMPDAY

Our daily backup procedure writes to magazine drive 1 on Mondays, Wednesdays, and Fridays and to magazine drive 2 on Tuesdays, Thursdays, and Saturdays. To make our night operator's job easier, we have him mount both magazine drives each night. Thus, we need a procedure that identifies the proper magazine for the current day of the week.

When the EVALUATE statements (see Figure 3-12) are included in the backup procedure, it computes a day of the week value in parameter 7, where a value of zero means Sunday, a value of one means Monday, and so on. The procedure then selects the appropriate magazine for the backup file, and the operator doesn't need to intervene.

Figure 3-12 *EVALUATE statements to determine day of the week. (This code appears as procedure CMPDAY on diskette.)*

```
// LOCAL BLANK-*ALL
// LOCAL OFFSET-257,DATA-'?DATE?'
*
// EVALUATE P1,2=?L'257,2'?
// EVALUATE P2,2=?L'259,2'?
// EVALUATE P3,2=?L'261,2'?
*
// IF ?1?>2  GOTO PSTFEB
// EVALUATE P1,2=?1?+12
// EVALUATE P3,2=?3?-1
*
// TAG PSTFEB
// EVALUATE P1,2=?1?+1
// EVALUATE P5,5=(?3?*365)+(?3?/4)
// EVALUATE P6,5=(?1?*30)+(?1?*6/10)
// EVALUATE P5,5=?6?+?5?
// EVALUATE P5,5=?5?+?2?
// EVALUATE P6,5=?5?/7
// EVALUATE P7,1=?5?-(?6?*7)
```

Computing Day of the Week in RPG

by Ed Antus

Code on diskette:
RPG code CMPDAY

Our present calendar, instituted in 1582 by Pope Gregory XIII, makes every fourth year a leap year except for centennial years, which are leap years only if evenly divisible by 400. (To correct for the extra leap years that had been added since the time of Julius Ceasar, Pope Gregory decreed that the date October 4, 1582, was to be followed by October 15, thus bringing the spring equinox back to March 21.) Because the Gregorian calendar has been in place since 1582, it is relatively easy to compute the day of the week for the first day of any month for any year since 1583 simply by determining the number of days that have gone by since January 1, 1583 (which happened to be a Saturday), and then dividing that number by seven and using the remainder to determine the day of the week.

The RPG E- and C-specs in Figure 3-13 perform these computations. The partial program assumes that field SYEAR contains the year and field SMON contains the month for which you want to know the day of the week for the first of the month. The partial program begins by calculating the number of days from January 1, 1583, until January 1 of the desired year as field TOTDYS. The Z-ADD operation sets on indicator 38 if there is no remainder (i.e., year is a leap year). Next, the program counts the number of century years and the number of quadracentennial years to correct for centuries that are and are not leap years. Finally, if the year is a leap year (as shown by indicator 38), and the month is less than March, that year's leap day is subtracted from field TOTDYS (the leap day doesn't affect the first day of January or February). The program then adds to field TOTDYS the number of days from January 1 to the first day of the desired month (as read from array DAT) and adds one more to get the number of days since January 1, 1583, for the first day of the desired month and year (field DSAUM). The division and remainder statements yield field WKDY1, the number of days into a new seven-day cycle. A table lookup based on field WKDY1 yields field FSTDAY, the number for the day of the week; a value of 1 indicates Sunday, a value of 2 indicates Monday, and so on.

Just for a historic note, the American colonies adopted the Gregorian Calendar in 1752 by "suppressing" the 11 days between September 2 and September 14, 1700 (a rule Benjamin Franklin thought would delight those who liked to sleep, for they could "lie down on the second of this month and not perhaps awake till the morning of the fourteenth"). Therefore, if you need to know on what day of the week William Penn's August 1684 mortgage payment was due, your answer will be in accordance with our calendar (new style), not the calendar he knew (old style).

Figure 3-13

Code to compute day of week. (This code appears as source member CMPDAY on diskette.)

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
     E                    DAT     12  32  3 0
     E                    TABDN    1   7  1 0 TABND    1 0
     .
     .
     .
     C*   SYEAR - century and year (1943, 2030 , 3501 , etc.)
     C*    SMON  - 2-digit valid month number 1-12
     C*
     C*   Calculate total days including extras for leap years.
     C            SYEAR     MULT 365.25    TOTDYS  92
     C                      Z-ADDTOTDYS    DADIFF  22      38
     C*
     C*   Determine number of century-years not leap years and
     C*   subtract from total days.
     C            SYEAR     DIV  100       NUMCNT  20
     C            SYEAR     DIV  400       CNTLPS  40
     C    38                MVR            CNDIFF  20 38
     C            NUMCNT    SUB  CNTLPS    NONLPS  20
     C *
     C                      SUB  NONLPS    TOTDYS
     C*
     C*   If data is in leap year, is earlier than March, and
     C*   extra day has not already been removed (century
```

```
C*   year divisible by 400), subtract 1 day.
C    38      SMON      COMP 3                      38
C    38                SUB  1        TOTDYS
C *
C            TOTDYS    ADD  DAT,SMON DASUM   70
C                      ADD  1        DASUM
C                      DIV  7        DAQUD   60
C                      MVR           WKDY1   10
C*
C*   FSTDAY - number of day of the week (1=Sunday, 2=Monday,
C*            Etc.)
C                      Z-ADD1        FSTDAY  10
C                      SETOF                     38
C            WKDY1     LOKUPTABDN    TABND         38
C    38                Z-ADDTABND    FSTDAY
C*


** DAT , TOTAL DAYS ARRAY/MONTH-TO-MONTH
000031059090120151181212243273304334
** TABON/TABND - FST DAY OF THE MONTH
07
11
22
33
44
55
66
```

Editing Fields Using O-Spec Edit Codes

by Mel Beckman

Code on diskette:
Assembler subroutine @DATA

Assembly language subroutine @DATA lets you use RPG I- and O-specs to perform data conversions (such as binary to decimal) without using a dummy disk file.

As an RPG programmer, I've often wanted to use the conversion and edit features of RPG I- and O-specs directly, without performing file I/O. For example, I may have a binary number in C-specs that I want to convert to decimal for use in some computations. Or I may want to edit a numeric field to insert commas and a decimal point and then store the edited result in an array element for display later as a list on the screen. It's easy, using RPG I-specs, to convert a binary number to decimal, or, using O-specs, to edit a dollar amount with decimals and commas. But it's impossible to use RPG's built-in conversions and edits "on demand" in RPG C-specs. Writing RPG calculations to perform such tasks is cumbersome, and the resulting routines run very slowly.

Fortunately, a simple assembly language subroutine can put all the data transformation capabilities of I- and O-specs on tap for use in your C-specs. The subroutine does this by exploiting an often-ignored feature of RPG: special device files. A special device file looks and works like a disk or printer file: you define the file using F-specs, access the file using RPG operation codes, and format data for output to and input from the file using O- and I-specs. Special device files, however, call a user-supplied assembly language routine to perform I/O instead of using the system-supplied devices such as disks, workstations, or printers.

@DATA is one such user-supplied routine whose output is directly connected to its input. Thus, whatever your RPG program outputs to the @DATA special device file, the program can read back in through the same file. By using the RPG EXCPT and READ operation codes, you can use the @DATA special device file from within C-specs to perform editing on output and conversion on input. Because no physical input or output takes place, the subroutine adds virtually no time to the execution of your program.

To use the subroutine, code an F-spec for the special device file as shown in Figure 3-13. In the device field of the F-spec (positions 40-46), code the word SRDATA to specify that @DATA is the assembler routine you're supplying to handle I/O for the special device file. (For special device files, RPG requires the user-supplied assembler routine name to be in the form @xxxx. On the F-spec, RPG requires you to replace the @ with SR.) The record length for the special file can be any value up to 4096 — whatever length you need to accommodate the data you want to transform. You also must supply I- and O-specs to carry out whatever data transformation you need; Figure 3-14 demonstrates editing a number using an RPG edit code and converting a binary number to decimal. Finally, whenever you want to perform a conversion in C-specs, code EXCPT and READ operations as shown in the example.

Using @DATA lets you transform any number of fields simultaneously in a single EXCPT/READ operation. You can gain further flexibility by using pseudo-record-ID constants on your O-specs and record-identifying logic on your I-specs to create any number of unique data transformation "sets."

Complete documentation for RPG special device files is contained in the S/36 *Programming with RPG* manual.

Figure 3-14

Example of RPG code using SRDATA routine

```
*      1         2         3         4         5  .      6      .  7         8
    FCONVERT UD  F     128           SPECIAL      SRDATA
    ICONVERT
    I                                      1  10 EDITED
    I                                 B   11  140DEC


    C                   EXCPTSRDATA
    C                   READ CONVERT


    OCONVERT E                  SRDATA
    O                           PLAIN M   10
    O                           BINARY    14
```

Re-creating Subroutine @DATA

If you don't have assembler subroutine @DATA, you can re-create it with procedure MK@DATA (you don't need IBM's Assembler Language Program Product to install @DATA). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MK@DATA. You need to run MK@DATA only once to create the @DATA subroutine.

```
// * 'Re-creating R-module @DATA in library #RPGLIB
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201 DATA-'00000000'     Number of $MAINT records
// LOCAL OFFSET-209 DATA-+
D97CC4C1E3C140404000000200000000000001020000300000001300000000000
// LOCAL OFFSET-273 DATA-+
000000000000000000803F000000000000000000C81000000
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new @DATA member to insert object code
// LOAD $FEFIX
// RUN
HDR  4BBD @DATA00000
PTF  8966 A@DATA.99 , #RPGLIB
DATA E169 00 0000 E2087CC4C7E3C140000[illegible]
DATA 8C1C 00 0020 0000000000000000000[illegible]
DATA C494 00 0040 E3370208634010208340203[illegible]
DATA T92F 00 0060 8801DC86021E3000002E3F2811[illegible]
DATA B885 00 0080 E331026802EB3F0102E3F1671[illegible]
DATA AA75 00 00A0 DFAC010C1EC2010000C2020000[illegible]
DATA 573A 00 00C0 C8FFFFE80000000040F15C[illegible]
DATA T846 00 00E0 000064008200000080000[illegible]
END  2A9D
```

Centering a String

by Edward L. Giron

Code on diskette.
RPG code CTRTXT

The code in Figure 3-15 centers a non-blank line of text regardless of its justification.

Figure 3-15

A routine to center text. (This code appears as member CTRTXT on diskette.)

```
*.... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
     E                    HDR      132  1

     C*
     C* Data structure redefining HDR array as the field HEADER
```

```
I*
I           DS
I                                     1 132 HDR
I                                     1 132 HEADER


I*
I* Subroutine to center the contents of array HDR
I*
C                    Z-ADD132       X
C          HDR,X     DOWEQ*BLANK
C                    SUB  1         X
C                    END
C*         X is the length of the input field.  Calculate
C*         the offset needed to center the text using
C*         formula
C*            StartPosition = (132 - FieldLength)/2 + 1
C          132       SUB  X         X
C                    DIV  2         X
C                    ADD  1         X
C*         Shift text to start in StartPosition
C                    MOVE *BLANKS   HDR
C                    MOVEAHEADER    HDR,X
C                    MOVEAHDR,1     HEADER
C                    END
C                    ENDSR
```

Justifying, Centering, and Converting Lowercase and Uppercase Strings

by Gary T. Kratzer

Code on diskette:
Assembler subroutines SUBRAT, SUBRCS

In *Searching for Strings*, I presented assembler subroutine SUBR$F, which performs a high-speed string search on a field. You could easily write subroutine SUBR$F in RPG or any other high-level language; however, its purpose is to give programmers a sort of "black box" routine that can perform this task much faster than a high-level language can. RPG's array processing logic is very slow when you reference an array with a variable index. I focus on array processing because programmers usually choose this method when they must perform string operations.

In this article, I also focus on RPG's lack of horsepower in this area by giving you two more assembler subroutines that perform string handling. By using these two routines, you can cut down on the overhead created by RPG array processing and thereby add some much needed horsepower to your programs. First, I provide subroutine SUBRAT, which left-justifies, right-justifies, or centers text within a field. And second, I offer subroutine SUBRCS, which converts text from uppercase to lowercase or vice versa.

East Side, West Side

To use subroutine SUBRAT in an RPG program, you must code an EXIT SUBRAT operation as follows:

```
C                    EXIT SUBRAT
C                    RLABL          OP       1
C                    RLABL          TEXT     ?
C                    RLABL          RCODE    1
```

• OP — a one-byte field that contains a code indicating the type of operation you want to perform. An L means left-justify, an R right-justify, and a C means center the text within the field. Note that when centering text, the possibility always exists that the text cannot be exactly centered — that is, one end may have one fewer blank than the other depending on the field size and number of characters to be centered. If this is the case, the left side of the text will have one fewer blank. For example, if subroutine SUBRAT centers the text NOW IS THE TIME in a 20-byte field, there are two blanks on the left and three on the right.

• TEXT — a field (no data structures allowed) up to 256 bytes long that contains the text to be adjusted. After returning from subroutine SUBRAT, the text is adjusted in this same field according to the operation you requested.

• RCODE — a one-byte field that contains the return code. This field will contain a 0 on a normal return or a 1 if the operation code was invalid.

Uptown, Downtown

Using subroutine SUBRCS in an RPG program is identical to using subroutine SUBRAT except for the first input parameter (OP) data. Again, you must code an EXIT SUBRCS operation as follows:

```
C                    EXIT SUBRCS
C                    RLABL          OP       1
C                    RLABL          TEXT     ?
C                    RLABL          RCODE    1
```

• OP — a one-byte field that contains a code indicating the type of operation you want to perform. An L means convert uppercase to lowercase, and a U means lowercase to uppercase. Note that the field to be converted can contain any mixture of alpha characters, numbers, or special characters, but only alpha characters are affected in the conversion. This way you can pass anything to subroutine SUBRCS, and only the characters that should be converted will be.

• TEXT — a field (no data structures allowed) up to 256 bytes long that contains the text to be converted. After returning from subroutine SUBRCS, the text is converted in this same field according to the operation you requested.

• RCODE — a one-byte field that contains the return code. This field contains a 0 on a normal return or a 1 if the operation code was invalid.

All Around the Town

Both subroutine SUBRAT and subroutine SUBRCS address common string handling problems programmers face in data processing. These subroutines are useful, for example, when merging leads files. Suppose you do telemarketing and trade lead lists with other companies. You want to merge their files with your own, but their standards for entering data differ from yours. They may let operators enter data free form, whereas yours must adhere to strict guidelines. They also allow upper- and lowercase names, and you allow only uppercase. You could use subroutine SUBRAT to left-justify the fields to remove leading blanks that inevitably would appear, and you could use subroutine SUBRCS to convert the data to all uppercase. These two subroutines can increase the efficiency of many of your programs.

Re-creating Subroutine SUBRAT

If you don't have assembler subroutine SUBRAT, you can re-create it with procedure MKSUBRAT (you don't need IBM's Assembler Language Program Product to install SUBRAT). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRAT. You need to run MKSUBRAT only once to create the SUBRAT subroutine.

```
// * 'Re-creating R-module SUBRAT in library @RPGLIB'
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000103'     Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'09E2E4C2D9C1E340400000000800000000000008000000000080000320000000002588'
// LOCAL OFFSET-273 DATA-+
'120718253100000080133300000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J BLOCKS-36,EXTEND-20
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-@RPGLIB
// END * Patch the new SUBRAT member to insert object code
// LOAD $FEFIX
// RUN
HDR , 3043 SUBRA00000
PTF , 0223 RSUBRAT,99,,@RPGLIB
DATA 0A16 00 0000 E2D9E2E4C2D9C1E3C00000003BA000000000000000000000000000000000000000
DATA 66DE 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 5857 00 0040 E2810031F2870FE2E4C2D9C1E340F149F04040404040340101273402018B2408
DATA 7784 00 0060 016F3C40037B0CFE0377037B0F01017801780F0101780031202827262110191 5
DATA 6778 00 0080 E32F0081017B35010I6F1C00017B031C00017B030C00005701780C000058917 8
DATA 299C 00 00A0 7B0109C2020179BC0000000B01014F7501023CF100002A3218191813010090 01
DATA 7128 00 00C0 E33900B701747003D0F2810F700900F3813B7DC300F281ECF287093CF00174C2
DATA E870 00 00E0 010179704000F201000F010178F282C4020101F18713C20202790036281E1401
DATA 8AC6 00 0100 E2810CC90C000DA7017B0C010DA600A79C000000F2878C3CF00174C2010107836
DATA 8C80 00 0120 01017870400DF201OE3F0101 78F2828E3701017 1F187002F281E1A1020090503
```

Continued

```
DATA 2B2A 00 01A0 [illegible]
DATA 8D41 00 0180 [illegible]
DATA 2421 00 0180 [illegible]
DATA 6EF5 00 01A0 [illegible]
DATA 18EF 00 01C0 [illegible]
DATA 8EEB 00 01E0 [illegible]
DATA 07C8 00 0200 [illegible]
DATA BDA0 00 0220 [illegible]
DATA F3BC 00 0240 [illegible]
DATA 4B81 00 0260 [illegible]
DATA 8489 00 0280 [illegible]
DATA 8480 00 0540 [illegible]
DATA 8188 00 0560 [illegible]
DATA 9185 00 0580 [illegible]
END 580C
```

Re-creating Subroutine SUBRCS

If you don't have assembler subroutine SUBRCS, you can re-create it with procedure MKSUBRCS (you don't need IBM's Assembler Language Program Product to install SUBRCS). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRCS. You need to run MKSUBRCS only once to create the SUBRCS subroutine.

```
// * 'Re-creating R-module SUBRCS to library #PPGLIB'
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'     Number of #MAINT records
// LOCAL OFFSET-209,DATA-+
'[illegible]'
// LOCAL OFFSET-273,DATA-+
'[illegible]'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J,BLOCKS-28,EXTEND-28
// RUN * Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-#PPGLIB
// END
* Patch the new SUBRCS member to insert object code
// LOAD $FCFIP
// RUN
[illegible]
[illegible]
DATA [illegible] 00 0000 [illegible]
DATA [illegible] 00 0020 [illegible]
DATA [illegible] 00 0040 [illegible]
DATA [illegible] 00 0060 [illegible]
DATA [illegible] 00 0080 [illegible]
DATA [illegible] 00 00A0 [illegible]
DATA [illegible] 00 00C0 [illegible]
DATA [illegible] 00 00E0 [illegible]
DATA [illegible] 00 0100 [illegible]
DATA [illegible] 00 0120 [illegible]
DATA [illegible] 00 0140 [illegible]
DATA [illegible] 00 0160 [illegible]
DATA [illegible] 00 0180 [illegible]
DATA [illegible] 00 01A0 [illegible]
DATA [illegible] 00 01C0 [illegible]
DATA [illegible] 00 01E0 [illegible]
END [illegible]
```

DFU, SDA, and SEU

CHAPTER

4

Preventing Member Naming Conflicts

by Ray W. DeMers

Code on diskette:
Procedure SEUMOD
Screen format member SEUMODFM

Have you ever inadvertently used the name of an SSP procedure when you created a new S/36 procedure in your user library? Then, when you tried to run the SSP procedure, your procedure ran instead — because when you run a procedure, the system looks in your current library before it searches #LIBRARY. To avoid this conflict, you should always check #LIBRARY before assigning a name to a new user procedure, but even the best of us sometimes forget to do that (after all, coding a new procedure is much more exciting than searching a directory listing). You can modify the IBM-supplied SEU procedure so that it will automatically notify SEU users of potential conflicts between the name of a user procedure that is being created under SEU and the name of any existing procedures in #LIBRARY. The modification requires only 15 lines of OCL. Because you will be modifying an IBM-supplied procedure, it is a good idea to make a backup copy of SEU in library #SEULIB.

Once you have the backup copy, add the new code in Figure 4-1 to the beginning of the S/36 SEU procedure. So that the added OCl statements can return messages to the SEU user, you also must create the screen format member in Figure 4-2a, which must be copied to library #SEULIB. The screen is shown in Figure 4-2b. These additional OCL statements cause SEU to run for an additional few seconds — a small price to pay for preventing name conflicts. However, if you find the time delay unacceptable, you can restore the original version of SEU using your backup copy.

Figure 4-1

Modifications to SEU procedure. (This code is contained in procedure SEUMOD on diskette.)

```
********************          Added code
*                             Check #LIBRARY for member presence
// IF ?5?/#LIBRARY      GOTO OK
// SWITCH 1111XXXX                                   . * 1-4 = NON-DISPLAY
// IF PROC-'?1?,#LIBRARY'     SWITCH 0XXXXXXXXX      . * PROCEDURE EXISTS
// IF SOURCE-'?1?,#LIBRARY'   SWITCH X0XXXXXXXX      . * SOURCE EXISTS
// IF LOAD-'?1?,#LIBRARY'     SWITCH XX0XXXXXXX      . * LOAD EXISTS
// IF SUBR-'?1?,#LIBRARY'     SWITCH XXX0XXXXXX      . * SUB-ROUTINE EXISTS
// IF SWITCH1-1  IF SWITCH2-1  IF SWITCH3-1   IF SWITCH4-1  GOTO OK
*
// TAG AGAIN
// PROMPT MEMBER-SEUMODFM,FORMAT-WERROR,LIBRARY-#SEULIB,UPSI-YES
// IF ?CD?/2001               GOTO OK                . * CMD-1  CONTINUE
// IF ?CD?/2007               CANCEL                   * CMD-7   CANCEL
// GOTO AGAIN
*
// TAG OK
*********************         End of added code
```

remainder of standard S/36 SEU procedure

Figure 4-2a

Screen format member SEUMODFM

```
* .     1         2         3         4         5         6         7         8
SWERROR    0124      YY                                        AG
D          790202Y               Y Y   Y                     WARNING
D   WARNING              WARNING                 WARNING
DMEMBER    080537Y                     Y Y
D          600710Y                   91            A PROCEDURE with this nX
Dame already exits in #LIBRARY!!
D          600810Y                   92            A SOURCE with this nameX
D already exits in #LIBRARY!!
D          600910Y                   93            A LOAD MEMBER with thisX
D name already exits in #LIBRARY!!
D          601010Y                   94            A SUB-ROUTINE with thisX
D name already exits in #LIBRARY!!
D          601310Y                                 This member is being usX
Ded to control system activities,
D          601410Y                                 therefore this Member-NX
Dame should not be used in any library
D          601510Y                                 except the System LibraX
Dry itself (#LIBRARY)!!
D          182330Y               Y                 CMD-7   End of job
D          392422Y               Y                 (CMD-1   Continue "SEU"X
D with this name)
```

Figure 4-2b

Screen format SEUMODFM for modified SEU procedure

```
          WARNING              WARNING              WARNING              WARNING

                                       ********

            A PROCEDURE with this name already exits in #LIBRARY!!
            A SOURCE with this name already exits in #LIBRARY!!
            A LOAD MEMBER with this name already exits in #LIBRARY!!
            A SUB-ROUTINE with this name already exits in #LIBRARY!!


            This member is being used to control system activities,
            therefore this Member-Name should not be used in any library
            except the System Library itself (#LIBRARY)!!





                                CMD-7   End of job
                        (CMD-1   Continue "SEU" with this name)
```

Printing Multiple Copies of DFU Reports

by Richard Comstock

On the S/36, you can place a // PRINTER statement before a // LOAD statement if you specify CONTINUE-YES. I've used the following code to obtain multiple copies of a DFU list:

```
// PRINTER CONTINUE-YES,COPIES-3
LIST DMM0150,DFUDMM,,,,,,,#DMASII
```

If you later want only a single copy of a particular printout produced, you can turn off multiple-copy printing by including a

```
// PRINTER CONTINUE-NO
```

statement before you request the single-copy printout.

Printing DFU Reports at 15 CPI

by John Blum

Before SSP Release 5.1, if you specified a value greater than 132 for printer line width on a DFU LIST procedure, the list would automatically print at 15 CPI. Alas, this is no longer true. An alternative is to use the SET procedure, but this approach is not satisfactory because the typical use of LIST is for a quick-and-dirty report.

I have solved the problem by adding three lines to the beginning of and modifying one line in IBM's #LIST procedure located in #DFULIB (Figure 4-3). This technique lets you use the SORT/NOSORT parameter as an indicator for 10/15 CPI. If you want 15 CPI and SORT, specify.PSORT for the SORT/NOSORT parameter; POSORT produces 15 CPI and NOSORT. The first EVALUATE statement defaults P64 to 10, so the procedure is not affected unless PSORT or POSORT is specified in the SORT/NOSORT parameter.

Figure 4-3

Modifications to IBM's #LIST procedure located in #DFULIB

```
// EVALUATE P64=10
// IF ?4?=POSORT EVALUATE P4='NSORT' P64=15
// IF ?4?=PSORT EVALUATE P4='SORT' P64=15
```

Also change the PRINTER statement to:

```
// PRINTER NAME-#DFPRINT,CPI-?64?
```

Changing Only Command Text in Menus

by Dennis Ruud

Code on diskette:
Procedure MCOM

Menus on the S/36 are slick, easy-to-build tools for running and keeping track of programs. SDA is the quickest, easiest way to build the menus, but sometimes you must change only a few little things in the command text. It is time-consuming to go through all the SDA screens and prompts and wait for the screen to recompile just for a missing comma or misspelled word. What you need is a quick way to change the command text without going through SDA.

A menu, screen format, and screen format member all have the same name (e.g., MYMENU). A command text source member bears the name of the menu with two pound signs appended (e.g., MYMENU##). To change only the command text of a menu, just use SEU, DSU, or FSEDIT to edit the command text. After making the changes, use the CREATE procedure to recompile the command text. Procedure MCOM in Figure 4-4 provides a quick way to edit and recompile the command text.

One word of caution. Don't serialize the command text when you end the editing session. If you serialize the command text, it will write over the menu name and option numbers. Also remember that the command text is a source member, not a procedure member.

Figure 4-4
Procedure MCOM

```
* USES SEU OR FSEDIT IN POP TO MAKE CHANGES TO COMMAND TEXT FROM
* A SPECIFIED MENU
*
* - LOCAL DATA AREA CONTENTS -
*   1 - 6 Menu Name
*   7 - 8 ## means we are after the command text of a menu
*
// LOCAL BLANK-*ALL,DATA-'?MENU?'   Load LDA with menu name
// LOCAL  OFFSET-7,DATA-'##'        ## characters attached to
*                                   menu name make it the command text
// IF ?L'1,6'?=  GOTO NOMENU        If 1st 6 characters are blank,
*                                   it's not a valid menu
*SEU ?L'1,8'?,S,?SLIB?              For SEU users
FSEDIT ?L'1,8'?,S,?SLIB?            For POP users
CREATE ?L'1,8'?,REPLACE,?SLIB?,HALT  Creates msg member from text
// GOTO END
// TAG NOMENU
// * 'REQUESTED MENU IS NOT A USER MENU  PROCEDURE TERMINATED '
// PAUSE
// TAG END
// LOCAL BLANK-*ALL
```

Diskettes

CHAPTER

5

Reading and Writing Diskettes from RPG

by Mel Beckman

Code on diskette:
Procedures RECVDK, SENDDK
RPG programs RECVDK, SENDDK
Assembler subroutine SUBRDK

In an example of generic tool design, the author shows how the program he designed to allow an RPG programmer to read and write any part of any diskette spawned another tool.

Although programmers are like other professionals in using specialized tools to practice their craft, they possess a unique ability. Unlike most workers who rely on tangible tools to extend their power or their reach, programmers seemingly are able to conjure their tools from thin air. This phenomenon seems so because programming tools are nothing more than programs themselves. Kernighan and Plauger, in their book *Software Tools*, suggest that to qualify as a truly useful tool, a program should be generic. For example, one can make a tool to read a disk file and print its contents on the printer, but a tool designed to copy data from any input device to any output device is much more useful. This more versatile tool can still print a file, by copying it from a disk to a printer. However, it also can copy from disk to disk, tape to printer, diskette to disk, and so on. A programming tool is made to be used, and its maker can justify the extra coding effort required to generalize it because a general-purpose tool normally gets more use than a special-purpose one. And another benefit comes along with the initial versatility: new tools can be created by building on top of existing tools.

In this spirit, I present two general-purpose tools (the second builds on the first) that let you access the diskette drive and send data directly from one S/36 diskette drive to another S/36 diskette drive. The first tool allows an RPG programmer to read and write any part of any diskette. As an example of its usefulness, one application I was working on especially needed this capability to sequentially read and write 2D diskettes. I could have written a standalone program in assembler language to provide only the capability I then needed. However, because I was creating a tool, I avoided assumptions about what others might want to do and created instead a small, general-purpose assembler subroutine that could read and write diskettes. The subroutine is designed to be called from RPG so that any RPG programmer can use it.

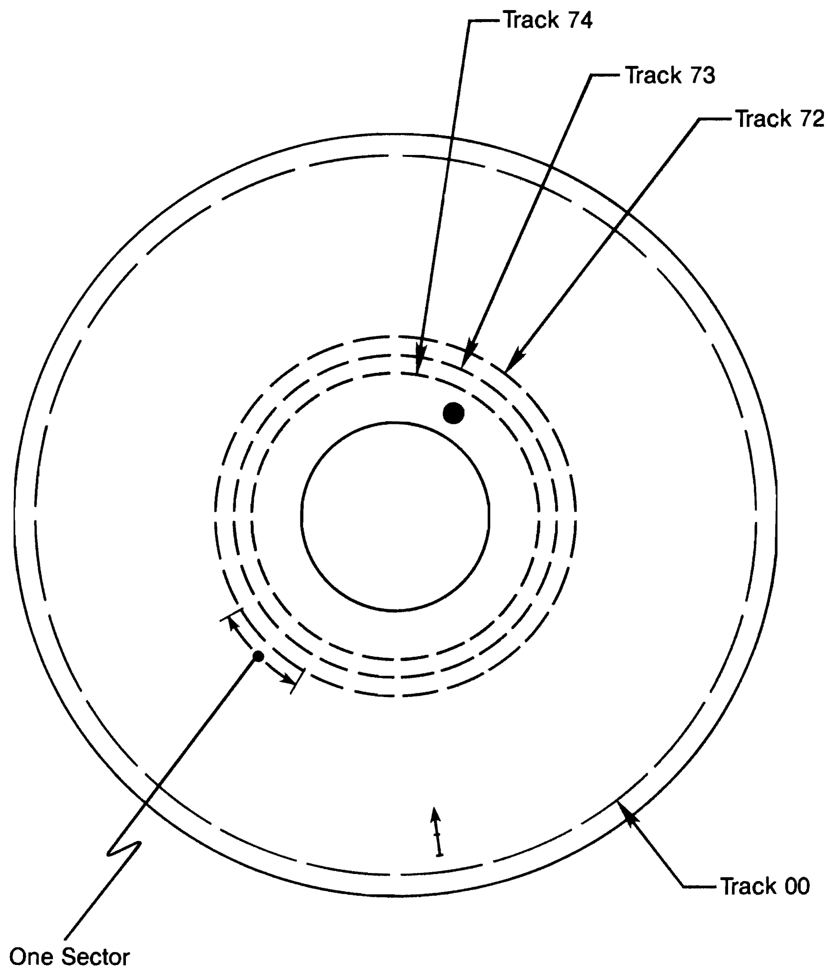
The application that originally needed the subroutine is a tool too — one that many S/36 users may find useful. This second tool is an RPG program that can copy a diskette from one machine (e.g., a S/36 5360) and write it directly onto a diskette in another machine (e.g., a S/36 PC) connected by a communications line. The fact that the source machine uses 8-inch diskettes and the target uses 5 1/4-inch diskettes is irrelevant — this is a general-purpose tool, remember? But I'm getting ahead of myself. Let's look at the first tool first.

The First Tool

Before I describe how to use subroutine SUBRDK, an assembler language subroutine that performs diskette I/O operations for an RPG program, a brief discussion of diskette anatomy will give us a common ground from which to proceed. Figure 5-1 illustrates the physical layout of a diskette. Every diskette contains 75 usable concentric circular tracks, numbered 00 through 74. Two-sided diskettes have another set of tracks on the second side. Each track is divided into sections, called sectors, that are analogous to records in a disk file. One sector is the smallest amount of data that can be read or written in one operation. The number of sectors per track and the number of bytes per sector determine the capacity of the diskette. Figure 5-2 summarizes the details of various diskette formats.

Figure 5-1

Physical layout of a diskette



Referring again to Figure 5-1, track 00 is called the index track because it contains the dataset labels for the files stored on the diskette. This is a kind of "table of contents" for the diskette (details about dataset labels can be found in the IBM manual *Diskette General Information* — GA21-9182). Regardless of how the rest of the diskette is initialized, the index track is always formatted to contain 26 128-byte sectors, each containing one dataset label. For two-sided diskettes, track 00 on the second side contains a continuation of the index track. It is formatted as 26 256-byte sectors, each containing two dataset labels.

Figure 5-2
Diskette format information

Diskette Type	How Initialized	Bytes per Sector	Sectors per Track	Bytes per Disket
1D	FORMAT	128	26	246,272
1D	FORMAT2	512	8	303,104
2D	FORMAT	256	26	985,088
2D	FORMAT2	1024	8	1,212,416

To call the subroutine from an RPG program, code an EXIT SUBRDK operation, which must be followed by a list of seven RLABL parameters (Figure 5-3). Each parameter is described below.

Figure 5-3
Calling sequence for subroutine SUBRDK

```
C     EXIT SUBRDK                      Call SUBRDK
C     RLABL          FUNC     1        Function code
C     RLABL          MOD      1        Modifier bits
C     RLABL          TRACK   20        Track number
C     RLABL          HEAD    10        Head number
C     RLABL          SECTOR  20        Sector number
C     RLABL          COUNT   20        Sector count
C     RLABL          BUFF              Buffer array
```

Function. This one-character field indicates the diskette operation to be performed. The codes are:

1: Read data. The number of sectors to be read is specified in the count parameter. You cannot read more sectors than your buffer can hold. Deleted sectors are bypassed (i.e., not counted).

2: Read data, including deleted sectors. Otherwise, this is identical to function code 1.

5: Write data. The number of sectors to be written is specified in the count parameter. You cannot write more sectors than your buffer can hold.

6: Delete. The number of sectors to be deleted is specified in the count parameter.

8: Select the diskette slot specified in the track parameter. Numbers 1 through 3 select the individual slots, 4 through 13 select the first magazine, and 14 through 24 select the second magazine.

9: Eject the diskette.

A: Orient the autoloader by positioning it at slot 1.

After the operation is completed, an error code may be returned in this field. If an error code is returned, the requested operation was not performed. The error codes are:

L: The buffer is too large — it cannot exceed 2,048 bytes.

S: The buffer is too small — it must be at least 256 bytes.

Modifier Byte. The modifier byte contains bits set to modify the operation being performed. You can set or clear the desired bits using the RPG operation codes BITON and BITOF. The bit numbers listed in Figure 5-4 are the ones to use in BITON or BITOF operations.

Figure 5-4
Bit numbers for BITON and BITOF

Bit 0 OFF	Single-density reading and writing.
Bit 0 ON	Double-density reading and writing.
Bit 4 OFF	One-sided diskette.
Bit 4 ON	Two-sided diskette.
Bit 6 OFF, 7 OFF	128-byte sectors
Bit 6 OFF, 7 ON	256-byte sectors
Bit 6 ON, 7 OFF	512-byte sectors
Bit 6 ON, 7 ON	1024-byte sectors

Track. This two-digit field specifies the number of the track to be read or written, from 10 to 74. For the select diskette function, this field contains the number of the slot to select. Remember that if you are reading the index track, you must specify single-density, 128-byte sectors for side one. For the side two-index track, specify double density, 256-byte sectors.

Head. This one-digit field specifies which read/write head to use. For one-sided diskettes, this field must be set to 0 (zero). For two-sided diskettes, a 0 (zero) indicates side one, and a 1 indicates side two.

Sector. This two-digit field specifies the number of the sector at which the read or write operation will start. More than one sector can be processed in one operation. Sectors are numbered from 1 to 26 for single-density recording, or 1 to 8 for double-density recording.

Count. This two-digit field specifies the number of sectors to read or write. You cannot process more sectors than will fit in the buffer.

Buffer. The buffer must be an array, from 256 to 2,048 bytes long. Only the total size of the array is significant. Code only the array name — don't use a field, data structure, or index name. Data will be read into or out of the array without regard to where individual array entries start or stop. For example, an array containing six 256-byte entries would constitute a buffer size of 1,536 bytes. If you were to read eight 128-byte sectors, the data would fill the first four 256-byte entries of the array (a total of 1,024 bytes).

Programs that use SUBRDK must allocate the diskette drive before the program is loaded by using the statement // ALLOCATE UNIT-I1. Failure to do this will cause the program to terminate abnormally. Also, whenever reading the index track, use the "Read Deleted" operation (code 2) because the index track almost certainly will contain some deleted sectors. If you encounter the message "Permanent Diskette I/O Errors" while debugging your own program that uses SUBRDK, you probably are trying to read or write the wrong sector size or density.

When I realized the breadth of the first tool's potential, my inclination to create tools of general usefulness emerged, and I designed a program that uses the first tool to copy a diskette from one S/36 to another. Hence, the result of applying the first tool produced a second tool in its own right, with its own special capabilities.

The Second Tool

Most models of the S/36 use 8-inch diskettes. The single exception is the S/36 PC (Model 5364), which uses only 5 1/4-inch diskettes. This presents a problem when data must be exchanged between the two machines. Somehow, one must be able to transfer files, libraries, and folders between the two machines — preferably by copying 8-inch diskettes directly onto 8-inch or 5 1/4-inch diskettes. IBM offers several solutions, each of which requires the purchase of between $700 and $1,800 of special IBM software, and possibly, depending on your system, an enhanced 5251 emulation board.

The best of the IBM solutions requires an IBM PC/AT directly attached to the 5364. The PC/AT method can copy a diskette in about 15 minutes and requires one operator intervention and an intermediate file on the PC/AT hard disk. But not everyone is likely to have a PC/AT handy because it costs nearly as much as the 5364 (many users have only a minimum-cost single-diskette IBM PC or compatible). If a PC/AT is not available, the copying process can require up to 45 minutes per diskette, depending on the technique used.

The solution presented here costs nothing, requires no operator intervention or special IBM software, and works with any kind of PC attached to a S/36 5364. The hardware requirements are modest: single-line communications on both S/36s and an inexpensive 9,600 bps (bits per second) modem eliminator (costs less than $200). Many users will have the communications feature installed already, making this a zero-cost alternative.

This technique depends on two not-so-obvious facts. First, the 8-inch and 5 1/4-inch diskettes, although different in size, are logically identical. That is, as far as the S/36 programming is concerned, both diskette sizes have the same internal format. Second, the 5364 is capable of transmitting data at 9,600 bps, even though IBM claims a limit of 4,800 bps. Why this is so isn't clear, but nothing in the IBM software or hardware prevents data transmission at 9,600 bps.

The idea here is to copy a 2D diskette from one machine directly onto a 2D diskette in another machine (the target machine) by passing the data over the communications line. Because no intermediate files are used to hold the diskette contents, no operator intervention is required, and the only time you need to be concerned with is the transmission time. At 9,600 bps, a diskette can be copied in 21 minutes.

The two RPG programs shown in Figures 5-5 and 5-6 implement the method. Both programs use subroutine SUBRDK to access the diskette drive directly. The first program, SENDDK, runs on the machine that contains the diskette to be copied. It reads the diskette directly and transmits the data over the communications line to the target machine, where it is received by the second program RECVDK. This union of the program and the subroutine achieved, RECVDK writes the data directly to the diskette as it is received. The procedures associated with each program are shown in Figures 5-7 and 5-8. Parameter 1 for each procedure is the magazine slot number to be selected, if any.

For simplicity, the programs accept slot numbers in the range of 01 through 24, just as SUBRDK expects them. If the machine doesn't have a magazine drive, parameter 1 should be left blank. Note also that if a magazine drive is installed, you must specify a slot number because slot 01 is not assumed. Running the programs establishes the communications link automatically, as long as Remote Workstation Support is not varied on.

A closer look at SENDDK reveals some interesting facets of the science of diskette copying. If a magazine slot is specified (i.e., passed in the LDA), SENDDK calls subroutine SUBRDK, which selects the diskette in that slot. Program SENDDK then reads and transmits the index track from side one of the diskette. Remember that the side one index track is formatted in single-density mode with 128-byte sectors for all diskette formats supported by the S/36. Only the last 19 of the 26 index sectors should be copied because the first seven sectors contain information specific to the physical layout of the diskette itself. Sector 7 contains the volume label, which you don't want to change, and other sectors below this contain the diskette bad-sector map, which is unique for each diskette. If this information were copied, problems could arise later when trying to read the new diskette. To avoid copying the information in the first seven sectors, program SENDDK begins reading at sector 8.

Next, the index track from side two is read and transmitted. The side two index track is formatted in double-density mode with 256-byte sectors.

Because SENDDK uses a 2,048-byte buffer, reading all 26 sectors requires four diskette operations. Finally, the 74 data tracks are read and transmitted. Each data track contains 16,384 bytes on both sides, so a total of eight diskette operations is needed. The RPG subroutine CPYTRK contains a small loop that accomplishes this task. The record length for the bisynchronous communications file is 2,048, so you can transfer the entire diskette buffer in one bisynchronous operation. The receiving program, RECVDK, is essentially a mirror image of program SENDDK — it receives diskette data from the communications line and writes it directly onto the diskette.

The programs, as currently written, copy all the tracks on a diskette, even if they do not all contain useful data. Thus, if a diskette is only half "full," 21 minutes are still needed for copying. To do otherwise would require that the programs analyze each dataset label to determine beginning and ending tracks — a process that would greatly complicate the programs while adding little to their utility.

The Hidden Benefits

After going through the process of implementing these two utilities, it's interesting to look at a few hidden benefits reaped by sticking to the generic tool philosophy. These programs are not restricted to transferring data between a 5360 and a 5364. They will copy diskettes from any S/36 model to any other S/36 model. And because the communications line is the medium, diskettes can be sent across town or across the country, with copying times ranging from four minutes (57,600 bps) to 42 minutes (4,800 bps). Because an exact duplicate of the diskette is being made, virtually any kind of 2D diskette can be copied (e.g., PTF, SSP). With minor program modifications, other diskette densities could be handled.

Because SUBRDK is a separate tool, new tools can be created by building on it in the same way SENDDK and RECVDK do. The possibilities are numerous. For example, it would be trivial to make a tool that reads a diskette into a temporary disk file and then copies that disk file any number of times to blank diskettes. This kind of mass diskette duplicator is something software distributors might find handy. For another example, consider users who must read I-Exchange diskettes created by the 5280 system, a programmable, intelligent workstation (no longer in production). They could grow their own utility to do this (the format is documented in publication GA21-9182, mentioned previously) and avoid having to buy IBM's feature 6000, which is necessary for S/36 users who want to read the 5280 diskettes. Enterprising readers will doubtless come up with their own tools built upon SUBRDK.

The purpose of this article has been twofold: to convey the concept of generalized programming tools and to illustrate some benefits of this concept through presentation of two genuinely useful utilities. Clearly, it makes sense to create tools with an eye toward future uses, even if those future uses are not immediately apparent. Distributing such tools to other programmers enhances the likelihood that the extra effort will pay of. I don't pretend to foresee all possible uses for the tools described in this article, but now other fertile minds are working on that problem.

Figure 5-5

Program SENDDK

```
*       1         2         3         4         5         6         7         8
H     064                                                                SENDDK
F*
F* TRANSMIT A DISKETTE VIA BSCA
F*
FCOMMOUT O   F20482048            BSCA
E                     BUFF     2048  1
TCOMMOUT ST EYM            REMOT1  REMOT2       97015
I*
I*— THE LOA CONTAINS THE DISKETTE SLOT TO BE SELECTEO, IF ANY
I*
I           UOS
I                                              1   2 SLOT#
C/EJECT
C*
C* IF A VALIO DISKETTE SLOT WAS PASSEO IN TRE LOA. SELECT THAT SLOT
C*
C           SLOT#     COMP '0''                       11  11 If slot# is
C   11      SLOT#     COMP '24'                         1111 between 0l and 24
C   11                MOVE SLOT#     TRACK                   Then set slot#
C   11                MOVE  8'       FUNC                    And
C   11                EXSR OKTIOS                            Select it
C*
C* READ ANO TRANSMIT TRE INDEX TRACK FROM SIDE 1
C*
C*   The index track on side one consists of 26 128-byte sectors
C*   recorded in single density mode   The first seven tracks contain
C*   physical diskette information that we don't want to copy. so we
C*   read sectors 8 through 20 and transmit them. then we read sectors
C*   21 through 26 and transmit those
C*
C*
C                     MOVE '2'       FUNC                   Read data/CAM
C                     BITOF'01234567'MOO                    Single density. 128
C                     Z-AODO         TRACK                  Track 0 is index trk
C                     Z-ADDO         READ                   Side 1
C                     Z-ADD8         SECTOR                 Start w/sector 08
C                     Z-A0013        COUNT                  13 sectors at once
C                     EXSR DKTIOS                           Read 1st part
C                     EXCPTCOMM                             Send it
C*
C                     Z-ADD21        SECTOR                 Continue w/sector 21
C                     Z-A006         COUNT                  6 sectors at once
C                     EXSR DKTIOS                           Read 2nd part
C                     EXCPTCOMM                             Send it
C*
C* READ AND TRANSMIT THE INOEX TRACK FROM SIDE 2
C*
C*   The index track on side two consists of 26 256-byte sectors
C*   recorded in double density mode   We read 8 sectors at a time and
C*   transmit them
C*
C                     BITON 07'       MOD                   256 byte sectors
C                     Z-ADD01         HEAD                  Side 2
C                     Z-A0001         SECTOR                Start w/sector 01
C                     Z-A0008         COUNT                 8 sectors per read
C                     EXSR DKTIOS                           Read 1st chunk
C                     EXCPTCOMM
C*
C                     Z-A0009         SECTOR                Continue w/sector 09
C                     EXSR DKTIOS                           Read 2nd chunk
C                     EXCPTCOMM                             Send it
C*
```

```
C                     Z-ADD17        SECTOR          Continue w/sector 17
C                     EXSR DKTIOS                    Read 3rd chunk
C                     EXCPTCOMM                      Send it
C*
C                     Z-ADD25        SECTOR          Continue w/sector 25
C                     Z-ADD02        COUNT           Only two left
C                     EXSR DKTIOS                    Read last chunk
C                     EXCPTCOMM                      Send it
C/EJECT
C*
C* READ AND WRITE THE 74 DATA TRACKS, BOTH SIDES
C*
C*   There are eight 1024-byte sectors on each track   The subroutine
C*   CPYTRK is called 74 times   It reads and transmits one track on
C*   each call
C*
C                     Z-ADD74       BEANS   20
C           LOOP      TAG
C                     EXSR CPYTRK
C                     SUB  1        BEANS      11
C    11               GOTO LOOP
C*
C* END OF JOB
C*
C                     SETON                    LR
C/SPACE 3
C*
C* COPY ONE TRACK - BOTH SIDES
C*
C           CPYTRK    BEGSR
C*
C                     BITON'067'     MOD             1024 byte sectors
C                     ADD  1         TRACK           Bump track number
C                     Z-ADD0         HEAD            Head 0
C                     Z-ADD1         SECTOR          Start with sector 1
C                     Z-ADD2         COUNT           2 sectors each time
C*
C* Read and transmit eight sectors from side 1, then eight sectors
C*   from side 2, two sectors at a time
C*
C           CPLOOP    TAG
C                     EXSR DKTIOS                    Read two sectors
C                     EXCPTCOMM                      Send them
C                     ADD  2         SECTOR          Bump sector number
C           SECTOR    COMP 8                   11    If done with side 1
C    11               Z-ADD1         SECTOR          Then start over
C    11               ADD  1         HEAD            With side two
C    11     HEAD      COMP 1                   11    If done with side 2
C    11               GOTO CPYEND                    Then return
C                     GOTO CPLOOP                    Else repeat
C*
C           CPYEND    ENDSR
C/EJECT
C*
C* DISKETTE IOS ROUTINE
C*
C           DKTIOS    BEGSR
C                     EXIT SUBRDK                    Call SUBRDK
C                     RLABL          FUNC    1       Function code
C                     RLABL          MOD     1       Modifier bits
C                     RLABL          TRACK  20       Track number
C                     RLABL          HEAD   20       Head number
C                     RLABL          SECTOR 20       Sector number
C                     RLABL          COUNT  20       Sector count
C                     RLABL          BUFF            Buffer array
C                     ENDSR
OCOMMOUT E                COMM
O                         BUFF     2048
```

Figure 5-6

Program RECVDK

```
*        1         2         3         4         5         6         7         8
H    064                                                                RECVDK
F*
F* RECEIVE A DISKETTE VIA BSCA
F*
FCOMMIN  ID  F20482048              BSCA
E                          BUFF     2048  1
TCOMMIN  SR EYA              REMOT2  REMOT1         97015
ICOMMIN  NS  01
I                                                  12048 BUFF
I*
I*- THE LDA CONTAINS THE DISKETTE SLOT TO BE SELECTED, IF ANY
I*
I            UDS
I                                                  1    2 SLOT#
C/EJECT
C*
C* IF A VALID DISKETTE SLOT WAS PASSED IN THE LDA, SELECT THAT SLOT
C*
C           SLOT#     COMP '01                      11  11 If slot# is
C   11      SLOT#     COMP '24                        1111 between 01 and 24
C   11                MOVE SLOT#     TRACK                  Then set slot#
C   11                MOVE  8'       FUNC                   And
C   11                EXSR DKTIOS                           Select it
C*
C* RECEIVE AND WRITE THE INDEX TRACK FROM SIDE 1
C*
C*   The index track on side one consists of 26 128-byte sectors,
C*   recorded in single density mode  The first seven tracks contain
C*   physical diskette information that we don't want to copy, so we
C*   receive sectors 8 through 20 and write them, then we receive sectors
C*   21 through 26 and write those
C*
C                     READ COMMIN                          Receive a buffer
C                     MOVE '5'       FUNC                  Write
C                     BITOF'01234567'MOD                   Single density, 128
C                     Z-ADD0         TRACK                 Track 0 is index trk
C                     Z-ADD0         HEAD                  Side 1
C                     Z-ADD8         SECTOR                Start w/sector 08
C                     Z-ADD13        COUNT                 13 sectors per receive
C                     EXSR DKTIOS                          Write 1st part
C*
C                     READ COMMIN                          Receive a buffer
C                     Z-ADD21        SECTOR                Continue w/sector 21
C                     Z-ADD6         COUNT                 6 sectors per receive
C                     EXSR DKTIOS                          Write 2nd part
C*
C* RECEIVE AND WRITE THE INDEX TRACK FROM SIDE 2
C*
C*   The index track on side two consists of 26 256-byte sectors,
C*   recorded in double density mode  We receive 8 sectors at a time and
C*   write them
C*
C                     READ COMMIN                          Receive a buffer
C                     BITON'07'      MOD                   256 byte sectors
C                     Z-ADD01        HEAD                  Side 2
C                     Z-ADD01        SECTOR                Start w/sector 01
C                     Z-ADD08        COUNT                 8 sectors per receive
C                     EXSR DKTIOS                          Write 1st chunk
C*
C                     READ COMMIN                          Receive a buffer
C                     Z-ADD09        SECTOR                Continue w/sector 09
C                     EXSR DKTIOS                          Write 2nd chunk
C*
C                     READ COMMIN                          Receive a buffer
C                     Z-ADD17        SECTOR                Continue w/sector 17
C                     EXSR DKTIOS                          Write 3rd chunk
C*
C                     READ COMMIN                          Receive a buffer
C                     Z-ADD25        SECTOR                Continue w/sector 25
C                     Z-ADD02        COUNT                 Only two left
C                     EXSR DKTIOS                          Write last chunk
C/EJECT
C*
C* RECEIVE AND WRITE THE 74 DATA TRACKS, BOTH SIDES
```

```
C*
C*   There are eight 1024-byte sectors on each track  The subroutine
C*   CPYTRK is called 74 times   It receives and writes one track on
C*   each call
C*
C                     Z-ADD74        8EANS   20
C           LOOP      TAG
C                     EXSR CPYTRK
C                     SU8  1         8EANS      11
C    11               GOTO LOOP
C*
C* END OF JOB
C*
C                     SETON                     LR
C/SPACE 3
C*
C* COPY ONE TRACK - 80TH SIDES
C*
C           CPYTRK    BEGSR
C*
C                     BITON'067'     MOD                1024 byte sectors
C                     ADD  1         TRACK              Bump track number
C                     Z-ADD0         HEAD               Head 0
C                     Z-ADD1         SECTOR             Start with sector 1
C                     Z-ADD2         COUNT              2 sectors each time
C*
C* Receive and write eight sectors on side 1, then eight sectors
C*   on side 2, two sectors at a time
C*
C           CPLOOP    TAG
C                     READ COMMIN                       Receive a buffer
C                     EXSR DKTIOS                       Write two sectors
C                     ADD  2         SECTOR             Bump sector number
C           SECTOR    COMP 8                    11      If done with side 1
C    11               Z-ADD1         SECTOR             Then start over
C    11               ADD  1         HEAD               With side two
C    11     HEAD      COMP 1                    11      If done with side 2
C    11               GOTO CPYEND                       Then return
C                     GOTO CPLOOP                       Else repeat
C*
C           CPYEND    ENDSR
C/EJECT
C*
C* DISKETTE IOS ROUTINE
C*
C           DKTIOS    BEGSR
C                     EXIT SUBRDK                       Call SUBRDK
C                     RLABL          FUNC    1          Function code
C                     RLABL          MOD     1          Modifier bits
C                     RLABL          TRACK   20         Track number
C                     RLABL          HEAD    20         Head number
C                     RLABL          SECTOR  20         Sector number
C                     RLABL          COUNT   20         Sector count
C                     RLABL          BUFF               Buffer array
C                     ENDSR
```

Figure 5-7

Procedure SENDDK

```
// * '*SENDDK* - SEND A DISKETTE VIA BSC   VOLUME  ?VOLID?'
*
* Parameter 1 is the diskette slot to select, if any
*
// EVALUATE P1,2-?1?                     Make Parm-1 two digits right justified
// LOCAL OFFSET-1,DATA-'?1?',BLANK-2     Put slot parameter in the LDA
*
// ALLOCATE UNIT-T1
// LOAD SENDDK
// COMM LINK-1
// RUN
```

Figure 5-8

Procedure RECVDK

```
[illegible]
```

Re-creating Subroutine SUBRDK

If you don't have assembler subroutine SUBRDK, you can re-create it with procedure MKSUBRDK (you don't need IBM's Assembler Language Program Product to install SUBRDK). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRDK. You need to run MKSUBRDK only once because SUBRDK is subsequently linked into programs SENDDK or RECVDK when they are compiled.

```
// * 'Re-creating R-module SUBRDK in library #RPGLIB'
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-301 DATA-'00000100'     Number of #MAINT records
// LOCAL OFFSET-709 DATA-+
'[illegible]'
// LOCAL OFFSET-773 DATA-+
'[illegible]'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J,BLOCKS-36,EXTEND-36
// RUN
* Copy created member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRDK member to insert object code
// LOAD #FCFIX
// RUN
[illegible]
[illegible]
DATA [illegible]
END [illegible]
```

Retrieving Deleted Diskette Files

by John B. Bowers

It has happened to every programmer, I suppose — that moment of nonchalance when you think you've finally become incapable of stupid mistakes. You do a group file delete without looking, and then . . . *Oops!*

Your blood turns to ice water, your throat constricts, and you fleetingly wonder whether the VW has enough gas to reach Mexico. Thank God for diskette backups, right? Well, no. That's where this nightmare begins.

The week before Christmas I was working late on the 5360, knocking out a year-to-date report using a group of temporary files. I finished in the early evening, spooled the report, and deleted the files — without looking. The files were grouped and subgrouped under AB.C.*nnn*. I should have deleted the group as AB.C; instead, I deleted it as AB. You can imagine the rest. AB.D.*nnn* and AB.E.*nnn* were live files, critical files, and they were gone in a flash.

Because I didn't realize my error in time, the real problem occurred the next day when Gretchen did the daily save on the AB files. As its first task, the backup she ran called the INIT procedure:

```
INIT WORK,,DELETE,M1.01,M1.10
```

Need I say more? I had deleted the files from F1, and now they were gone from I1 as well. I learned the cheerful news when I called in to let the office know I was planning to take the day off. A few terse words and I was in the car, wondering about Mexico and trying to remember everything I had ever heard about the PATCH procedure, which was not much.

I was reasonably certain of one thing — the data should still be on the diskettes. INIT ,,DELETE does not erase a diskette; it merely deletes the volume table of contents (VTOC). If I could somehow rebuild that VTOC, I should be able to restore those files.

It took several fruitless phone calls and two days of cold sweat to get the job done. The manual in which the PATCH procedure is documented is difficult to obtain (*Program Problem Diagnosis and Diagnostic Aids*, LY21-0590), and on-line Help isn't much help. Using, of all things, a booklet the IRS puts out about magnetic media W2s, I managed to decipher the header record layouts for the deleted files and rebuild their VTOC entries with the PATCH procedure. Just a month later, using the same PATCH techniques on a 5364, I was able to recover for a client files on the little 5 1/4 inch diskettes, as well. So the same procedures apply.

Can You Recover?

Before I describe how I rescued files with the PATCH procedure, I need to say a few words about deletions. A diskette file can get smoked in two primary ways: with the DELETE command or with the INIT command. Each method has options — some allow recovery, and some don't.

The DELETE command offers three types of delete: SCRATCH, ERASE, and REMOVE. The default is SCRATCH, and it is also the safest. When you run DELETE with the SCRATCH option, the computer sets the expiration date to the current session date. Both the data and the VTOC remain intact until another file is written onto the diskette. The ERASE option literally erases the data from the data tracks, leaving the file without recovery. The REMOVE option obliterates the VTOC but leaves the data intact, allowing recovery of the header records via the methods described here.

The INIT procedure likewise has options: FORMAT and FORMAT2 initialize the entire diskette, leaving no data anywhere to be recovered, whereas the RENAME option (the default) changes the volume ID and owner ID but does not affect labels or data. The DELETE option is the same as DELETE ALL,I1,REMOVE; it deletes only the label record but leaves data in a recoverable state.

Perhaps the following explanation of how to recover those missing files that are recoverable will be helpful to you — if you ever get to be as cocky as I was.

Beginning the Recovery Process

If you have password security, you can run PATCH from any display station. If not, you must work from the console. Only users with service aid authority can use PATCH. With the diskette to be patched in slot 1, you begin by typing the command PATCH I1 from any menu or command screen. You can also access PATCH through the HELP menu. Select Option 8, which brings up the PROBSERVE menu. Now select Option 2, and you have the SERVICE menu. Option 10 on this menu is the PATCH procedure. Be sure to specify I1, the diskette drive, because PATCH defaults to F1, the hard disk. Only the very brave should fool around with F1. The PATCH diskette utility setup screen (Figure 5-9) calls for a diskette address in one of three formats: sequential sector address; cylinder, head, record address; and label sector address.

Figure 5-1 on page 79 illustrates the physical layout of a diskette. A diskette contains 75 usable concentric circular tracks numbered 00 through 74. Two-sided diskettes have another set of tracks on the flip side. Each track is divided into sectors, which are analogous to records in a disk file. One sector is the smallest amount of data that can be read or written in one operation. The number of sectors per track and the number of bytes per sector determine the capacity of a diskette.

Track 00 is called the index track because it contains dataset labels for files stored on the diskette — the VTOC for the diskette. Regardless of how the rest of the diskette is initialized, the index track is always formatted to contain 26 128-byte sectors, each containing one dataset label. For two-sided diskettes, track 00 on the flip side contains a continuation of the index track, formatted as 26 256-byte sectors. (For more information about diskette internal formats, see the *IBM Diskette General Information Manual*, GA21-9182.)

To rebuild the VTOC on Track 00, you first look at label sector address 0000007L where the VOL1 (i.e., volume label) record resides. (The first six tracks, usually referred to as the CE tracks, are used for diagnostics.) To look at the volume label record, you must enter the address as shown in Figure 5-10. Because this sector is a label sector, you need to enter an L where the S is, to the left of HEX in the last line. Also, do yourself a favor and eliminate the need to do hex-to-decimal conversion gymnastics by changing the word HEX (the default) to DEC on the last line.

Sector address 0000007L (Figure 5-11a) shows the volume label

(WORK) and owner ID (EXPEDATA) of the diskette. The screen looks like a disk dump, with four columns of hex values on the left and the corresponding EBCDIC values on the right. When you begin patching, you enter hex values on the left for the missing header records. How accurate you are can be seen on the right when you press Enter. I advise you to check your progress periodically.

Each large column on the left contains four hex bytes per row; two positions, or nibbles, constitute a byte. The entire screen constitutes a 128-byte record. You must keep track of the hex positions yourself. It's a bit difficult at first, but as soon as you get used to multiplying everything by 16, it's easier. For example, the fifth row from the top starts in position (1 + (16 * 4)), or position 65, and ends in position (16 + (16 * 4)), or position 80. From this screen, you can move forward or backward using Command keys. Command key 1 pages to the next sector (0000008L), and Command key 2 pages to the previous label sector (0000006L). The actual work will begin in label sector 0000008L (where the label records begin), so use Command key 1 to page forward.

A newly initialized diskette looks like Figure 5-11b. If you have deleted a VTOC entry, the screen will look like Figure 5-11c. This screen is a dead giveaway that everything has been nuked; note the words "deleted sector," the D in the first position, and all the hex blank (i.e., '40') values. Paging forward from deleted sectors, you may find labels that look like Figure 5-11d — if there are more files on the diskette that haven't been deleted. Be careful to skip over them as you patch. For each file you recover, you must change a label that looks like the screen in Figure 5-11b or Figure 5-11c to a label that looks like the screen in Figure 5-11b. Figure 5-12 details a diskette label record layout. To make your files restorable, each field must be rebuilt correctly.

The old saying goes that there are eight ways to stick a diskette into a computer, and seven are wrong. The same complexity comes into play when you rebuild the label records on the index track. Are you dealing with files from a SAVE, SAVELIBR, SAVEFLDR, ARCHIVE, TRANSFER, or FROMLIBR operation? If the files came from a SAVE, were they saved as a group? Compressed? Multivolume? What was the record length? Is the diskette 1D or 2D? Was the diskette initialized using FORMAT or FORMAT2? You must know this information.

You have two primary ways to find out what kind of data lives on the diskette and where it is located. One is to refer to a diskette catalog that was printed before your disaster, which makes the job monumentally easier. The other is to plow through the entire diskette one sector at a time using Command key 1. For the sake of this exercise, assume you are working with 2D diskettes initialized as FORMAT2. (The chart in Figure 5-13 compares diskettes and formats.)

Finding the Missing Label Information

By paging forward through the label sector addresses, you have identified the label sectors you need to recover. To rebuild the missing label records, you must know where the actual data resides. The physical data begins in sector 1 of Track 01. To get to Track 01, enter the value 0000001 (i.e., the sequential sector address for sector 1) at the SS@ prompt on any PATCH screen, and blank out the next field (which contains an L if you are in Track 00). Press Enter, and you find yourself looking at the first sector of Track 01 (Figure 5-14a). For 2D diskettes initialized as FORMAT2, each sector contains 1,024 bytes. You can page through the sector 256 bytes at a time using the Roll keys, but to get to the next sector you must use Command key 1.

In this example, each of the 74 tracks has eight 1,024-byte sectors. It takes a long time to page through that many sectors using Command key 1! Yet, if you don't know what files are on the diskette, or where they are located, that is what you must do. Enjoy.

Figure 5-14a shows what the beginning of a data file looks like. The clue is FMT1 (embedded format 1) in position 1 of the record. The file name (in this example, LH.P.510) begins in position 8 and the file date (in YYMMDD format) in position 16. #SAVE (the default) is the set name assigned to the files when they were saved, and the file name in position 83 is the name of the file that follows if there was a group save (in this case, LH.P.206).

Figure 5-14b details what the start of a FROMLIBR record looks like. The S in position 1 shows this record to be a source member. The member name starts in position 2. FROMLIBR on the S/36 always has a record length of 008.

Figure 5-14c is an example of a library saved with SAVELIBR. Unless you knew the library record was there, I'm not sure you could recognize it. The library name starts near the bottom of the screen, in position 205 (#WORK). SAVELIBR always has a record length of 128.

Figure 5-14d is an example of a TRANSFER file. TRANSFER records are found only on diskettes initialized as FORMAT because EXCHANGE format does not work under FORMAT2.

Rebuilding the Label Records

Before you begin rebuilding the label records, take another look at Figure 5-14a. At the top of the screen is the sector address. The address following the SS@ prompt is the same location on diskette that you would see under FILE LOCATION on a catalog printout (i.e., sequential sector address in Figure 5-9). The address following the CHR prompt is the cylinder (i.e., track), head, record (i.e., sector) address. The CHR address format is the address format you use in building the label records — *if* you changed from HEX to DEC mode when you started PATCH I1. If you page forward through a few sectors, you notice that each track (cylinder) has eight sectors and two heads (00 and 01). Address 010008 gives way to 010101, and address

010108 to 020001. The last address on the diskette in our example is 740108. The format is CCHHSS (e.g., 740108 means track 74, head 01, sector 08).

With this very primitive background in what data looks like on the diskette, let's begin rebuilding the label for file name LH.P.510. The first data sector for this file is shown in Figure 5-14a. The label, when you finish, will look like Figure 5-11d. Step by step, let's walk through the fields in the label record and understand them.

To begin the patch, you move the cursor to the appropriate position in the hex data area (*not* the character data area on the right) and type in the replacement values (see Figure 5-15 for a table of EBCDIC hex values). This step is difficult and demands patience. It's best to address the fields in the label record one at a time and press Enter (without the P Command code) to check your work. You can avoid looking up hex codes for EBCDIC character values and instead key these characters directly by preceding each with a single quote character. Thus, you can key "FRED" as 'F'R'E'D and the file name as 'L'H'.'P'.'5'10.

Positions 1 through 4 contain the constant value HDR1.

Positions 6 through 13 contain the file name.

Position 5 and Positions 14 through 22 are reserved. Ignore them.

Positions 23 through 27, the diskette record length, are 01024 if the diskette was initialized as FORMAT2 or 00256 if initialized as FORMAT. TRANSFER files are always 00128.

Position 28 is a constant: R for files, blank for TRANSFER.

Positions 29 through 33 hold the beginning address of the file in CCHSS format. Remember the address on the right of the data sector following the CHR prompt? Just drop the leading zero of the cylinder — 00 becomes 0, 01 becomes 1 (e.g., address 010001 becomes 01001; 020107 becomes 02107).

Position 34 is a constant: 3 if FORMAT2, 1 if FORMAT.

Positions 35 through 39 contain the ending address of the file. This address is the CCHHSS address found on the last sector of data in the file. If you omit this entry, a restore of the file either won't work or will give unpredictable results. To find this address, locate the next file after the file on which you are working, and use Command key 2 to look at the sector just preceding. That address gives you the sector address of the end of the current file. If there is no next file, that's a problem. If the diskette was initialized before the SAVE, you probably can detect whether the file is the last file because it will be followed by a bunch of hex initialization values (see Figure 5-16). Changes in data patterns, such as character data to packed numbers, may also be a tip off to an end of file. Finally, you can always resort to trial-and-error and check your guess by browsing the results using POPLIB.

Position 43 is a constant: P if this file was saved in COMPRESS format, otherwise blank.

Position 44 is a constant: E for data, folder, or library file, H for TRANSFER.

Position 45 is used for multivolume files only: C indicates whether this file continues on the next diskette; L indicates the last installment of a multivolume file.

Positions 46 through 47 are also for multivolume files only and contain the sequence number of the current volume. For example, a file spanning three diskettes would appear as C01 in positions 45 through 47 of the first diskette, C02 on the second diskette, and L03 on the last diskette.

Positions 48 through 53 indicate the date the file was created. This date is optional, but you should include it, if possible, because diskette expiration dates are computed using this date as a starting point. You can use any date because the computer doesn't really care when the file was built. The date format is YYMMDD.

Positions 54 through 57 contain the record length of the actual data. *Data files will not restore if this value is incorrect.* Find this number on the diskette catalog or from some other source such as a data dictionary or program listing. Sometimes you can deduce the length if you are working with consistent data (e.g., alphabetized customer names). Note the following defaults:

```
TRANSFER = 0128
SAVELIBR = 0128
FROMLIBR = 0008
SAVEFLDR = 2560
```

Positions 67 through 72 hold the expiration date of the file in YYMMDD format. If this file is protected (i.e., retention 999 on the SAVE command), you enter 999999. Otherwise, use the expiration date of your choice.

Positions 75 through 79 indicate the sector address of the file following this one, if any. This field is required, and reconstructing it is usually as simple as adding 1 to the ending sector address. (But keep in mind that CHR addresses "roll over" after sector 08 — e.g., 010008 becomes 010101.) For multivolume files that are continued on the next diskette, this address is 75001.

Positions 96 through 106 are the constant value IBMSYSTEM36.

Position 109 is blank unless this file was part of a group save, in which case you use a constant value 1.

Position 110 contains a constant value:

```
    1 = data file
    2 = FROMLIBR or ARCHIVE
    3 = SAVEFLDR
    4 = SAVEFLDR extent
    9 = SAVELIBR
blank = TRANSFER.
```

Remember that the first 26 sectors are 128-byte records (of which only the last 19 — sectors 08 through 26 — are available for label records). Beginning with sector 27 on a two-sided diskette, the sectors become 256-byte records. You can record two labels on each of these 52 additional sectors, but for the positions of the second record, you must add 128 to the position numbers above.

When all label positions in a record are restored, you must move the cursor to the L at the top of the screen (to the left of CHR) and *replace that L with a P* — for PATCH. (If you neglect to change the L to P, none of your changes will be applied permanently. You then press Enter to apply the changes. A reassuring message should appear in reverse image at the lower right of the screen: "Sector is patched."

The PATCH utility is a handy parachute. Don't be afraid to use it when appropriate. Better yet, don't ever let yourself become so perfect at your craft that you do things without looking, else you may find yourself writing the next how-to article for *NEWS 3X/400.*

Figure 5-9
PATCH utility screen

```
                    S/36 PATCH DISKETTE UTILITY                       W2

Select Diskette sector(s)

Reply formats   SSSSSS  S        Diskette sequential sector address

                CCHHSS  C        Diskette cylinder head record address

                SSSSSS  L        Diskette label sector address

                        E        Exit option

                        S        HEX (HEX,DEC)




Cmd7-End
```

Figure 5-10
PATCH utility screen with entry

```
                    S/36 PATCH DISKETTE UTILITY                         W2

Select Diskette sector(s)

Reply formats   SSSSSS  S        Diskette sequential sector address

                CCHHSS  C        Diskette cylinder head record address

                SSSSSS  L        Diskette label sector address

                        E        Exit option

                000007  L        DEC (HEX,DEC)





Cmd7-End
```

Figure 5-11a
Volume label record

```
                    S/36 PATCH DISKETTE UTILITY                         W2

SS@- 0000007  L CHR- 000007    Decimal
    Addr    00        04        08        0C
    0000    E5D6D3F1  E6D6D9D2  40404040  40404040    *VOL1WORK        *
    0010    40404040  40404040  C9C2D4E2  E8E2E3C5    *        IBMSYSTE*
    0020    D4F3F640  40C5E7D7  C5C4C1E3  C1404040    *M36  EXPEDATA   *
    0030    40404040  40404040  40404040  40404040    *                *
    0040    40404040  404040D4  404040F3  404040E6    *       M   3   W*
    0050    40404040  40404040  40404040  40404040    *                *
    0060    40404040  40404040  40404040  40404040    *                *
    0070    40404040  40404040  40404040  40404040    *                *






Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-11b

Initialized label record

```
                       S/36 PATCH DISKETTE UTILITY
W2

 SS@= 0000008  L CHR= 000008   Decimal
     Addr    00        04        08        0C
     0000    C8C4D9F1  40C4C1E3  C1404040  40404040    *HDR1 DATA       *
     0010    40404040  4040F0F0  F0F8F040  F0F1F0F0    *      00080 0100*
     0020    F1F3F7F4  F1F0F840  404040C5  40404040    *1374108    E    *
     0030    40404040  40404040  40404040  40404040    *                *
     0040    40404040  40404040  4040F0F1  F0F0F140    *          01001 *
     0050    40404040  40404040  40404040  40404040    *                *
     0060    40404040  40404040  40404040  40404040    *                *
     0070    40404040  40404040  40404040  40404040    *                *

 Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-11c

Deleted label record

```
                       S/36 PATCH DISKETTE UTILITY                      W2

SS@= 0000009  L CHR= 000009   Decimal             Deleted sector
  Addr    00        04        08        0C
  0000    C4404040  40404040  40404040  40404040    *D               *
  0010    40404040  40404040  40404040  40404040    *                *
  0020    40404040  40404040  40404040  40404040    *                *
  0030    40404040  40404040  40404040  40404040    *                *
  0040    40404040  40404040  40404040  40404040    *                *
  0050    40404040  40404040  40404040  40404040    *                *
  0060    40404040  40404040  40404040  40404040    *                *
  0070    40404040  40404040  40404040  40404040    *                *

Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-11d

Rebuilt label record

```
                    S/36 PATCH DISKETTE UTILITY                              W2

SS@= 0000008  L CHR= 000008   Decimal
   Addr     00        04        08        0C
   0000     C8C4D9F1  40D3C84B  D74BF5F1  F0404040    *HDR1 LH.P.510   *
   0010     40404040  4040F0F1  F0F2F4D9  F0F1F0F0    *       01024R0100*
   0020     F1F3F0F1  F0F0F240  4040D7C5  404040F9    *1301002    PE   9*
   0030     F0F0F2F0  F7F0F2F0  F0404040  40404040    *002070200        *
   0040     4040F9F0  F0F2F0F8  4040F0F1  F0F0F340    *  900208  01003 *
   0050     40404040  40404040  40404040  404040C9    *               I*
   0060     C2D4E2E8  E2E3C5D4  F3F64040  F1F14040    *BMSYSTEM36  11   *
   0070     40404040  40404040  40404040  40404040    *                 *

Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-12

Diskette label record layout

Position	Description	Position	Description
1-4	HDR1	46-47	Sequence number (multivolume only)
6-13	File Name	48-53	File date
23-27	Diskette Record Length	54-57	Record length
28	R (blank for TRANSFER)	67-72	Expiration date
29-33	Starting address (CCHSS)	75-79	Start address of next file
34	1 (FORMAT) 3 (FORMAT2)	96-106	IBMSYSTEM36
35-39	Ending address (CCHSS)	109	1 = group save else blank
43	P= compressed format	110	1 = data file 2 = FROMLIBR or ARCHIVE 3 = SAVEFLDR 4 = SAVEFLDR extent 9 = SAVELIBR blank = TRANSFER
44	E= file, folder, library H= TRANSFER file		
45	C= continued on next volume L= last volume of file else blank		

Figure 5-13

Diskette size basics

Diskette Type/Format	Record Size	Ending Address
1D FORMAT	00080	73026
1D FORMAT2	00256	74015
2D FORMAT	00256	74126
2D FORMAT2	01024	74108

Figure 5-14a

Beginning of a file

```
                        S/36 PATCH DISKETTE UTILITY                          W2

  SS@- 0000001    CHR- 010001   Decimal
     Addr    00        04        08        0C
     0000    C6D4E3F1  000000D3  C84BD74B  F5F1F0F9    *FMT1 ..LH.P.5109*
     0010    F0F0F2F0  F7E30080  00C81C00  1C000000    *00207T.0.H......*
     0020    00000000  00000000  04000000  0008C1D7    *.. ....... .. .AP*
     0030    C9D5E5D9  404013D2  3C75530E  00A00001    *INVR   K.e..u. *
     0040    E7610000  00200003  7BE2C1E5  C5404040    *X/. . ..#SAVE   *
     0050    01C3D3C8  4BD74BF2  F0F6007B  D4C3E2C4    *.CLH.P.206.#MCSD*
     0060    C3E34000  00000000  00000000  00000000    *CT  ..        .. ..*
     0070    00000000  00000000  00000000  00000000    * ......      ..  .*
     0080    00000000  00000000  00000000  00000000    *.. ...       .  .  *
     0090    00000000  00000000  00000000  00000000    *... .  .     .... .*
     00A0    00000000  00000000  00000000  00000000    * ..... ..          .*
     00B0    00000000  00000000  00000000  00000000    *... ...      ... .  *
     00C0    00000000  00000000  00000000  00000000    *    .   .      .  .*
     00D0    00000000  00000000  00000000  00000000    *    .   .  .  .  .*
     00E0    00000000  00000000  00000000  00000000    *     . ..    ..  .*
     00F0    00000000  00000000  00000000  00000000    *      .  .  ..  . *



Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-14b

FROMLIBR record

```
                        S/36 PATCH DISKETTE UTILITY                          W2

  SS@- 0000117    CHR- 080005   Decimal
     Addr    00        04        08        0C
     0000    E2C1D7C3  C8C5C3D2  4000000C  60002C00    *SAPCHECK  ..-  .*
     0010    00000000  00000051  00062000  00000289    *       .e.    .. i*
     0020    01181125  40000000  80551800  00000000    *.      ...0i .    *
     0030    00000000  00000000  CAF0F0F0  F140C65C    *     .   0001 F**
     0040    5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C    ******************
     0050    5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C    ******************
     0060    5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C    ******************
     0070    5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C    ******************
     0080    5C5C5C16  87F0F0F0  F240C65C  5987F0F0    ****.g0002 F*ßg00*
     0090    F0F340C6  5C1B897D  C1D7C3C8  C5C3D27D    *03 F*.i'APCHECK'*
     00A0    3587F0F0  F0F440C6  5C59B1F0  F0F0F540    *.g0004 F*ß£0005 *
     00B0    C65C40D9  C5C3D6D9  C440D3C1  E8D6E4E3    *F* RECORD LAYOUT*
     00C0    40C6D6D9  40E3C8C5  40C161D7  40C3C8C5    * FOR THE A/P CHE*
     00D0    C3D240D9  C5C7C9E2  E3C5D94B  2F87F0F0    *CK REGISTER..g00*
     00E0    F0F640C6  5C59CAF0  F0F0F740  C65C5C5C    *06 F*ß0007 F****
     00F0    5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C    ******************



Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-14c

SAVELIBR library record

```
                      S/36 PATCH DISKETTE UTILITY                            W2
SS@- 0000949    CHR- 600005   Decimal
     Addr      00        04        08        0C
     0000      0003AA5D  03AD0E03  AA5E03AA  61001000     *. i)._..i. i/   *
     0010      03000000  0003AA62  03AD0E02  A903AD0B     *       i_._..z._.*
     0020      00040000  00000000  00000000  00000000     *             .   *
     0030      00000000  00000000  00000000  00000000     *  .        ..  . *
     0040      00000000  00000000  00000000  00000000     *         .     . *
     0050      00000000  00000000  00000000  00000000     *         .   .   *
     0060      00000000  00000000  00000000  00000000     *           .  .  *
     0070      00000000  00000000  00000000  00000000     *.     . . .      *
     0080      00000000  00000000  00000000  00000000     *        ..     . *
     0090      00000000  00000000  00000000  00000000     *  .              *
     00A0      00000000  00000000  00000000  00000000     *           . .   *
     00B0      00000000  00000000  00000000  00000000     * .     . .  .  .*
     00C0      00040002  A9409002  07000006  7BE6D6D9     *.   z °   .  #WOR*
     00D0      D2404040  00000500  02B20500  010010E2     *K    .    ¥  . .S*
     00E0      00000000  00000000  00000000  00000000     *  . .  .   .  .  *
     00F0      00000000  00000000  00000000  00000000     *   .       ...   *



Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Figure 5-14d

TRANSFER file

```
                      S/36 PATCH DISKETTE UTILITY                            W2

  SS@- 0000001    CHR- 010001    Decimal
     Addr      00        04        08        0C
     0000      F1C1F1F9  F8F94040  40404040  40404040     *1A1989          *
     0010      40404040  40404040  E8D6E4D9  40C3D6D4     *        YOUR COM*
     0020      D7C1D5E8  40404040  40404040  40404040     *PANY            *
     0030      40404040  40404040  40404040  40404040     *                *
     0040      40404040  40404040  4040C6C9  D9E2E340     *          FIRST *
     0050      C1C4C4D9  C5E2E240  D3C9D5C5  4B4B4B4B     *ADDRESS LINE... *
     0060      4B4B4B4B  4B4B4B40  40404040  40404040     *                *
     0070      40404040  40404040  40404040  40404040     *                *
     0080      00000000  00000000  00000000  00000000     *                *
     0090      00000000  00000000  00000000  00000000     *        .       *
     00A0      00000000  00000000  00000000  00000000     *          . . . *
     00B0      00000000  00000000  00000000  00000000     *                *
     00C0      00000000  00000000  00000000  00000000     *.             . *
     00D0      00000000  00000000  00000000  00000000     *        .       *
     00E0      00000000  00000000  00000000  00000000     *                *
     00F0      00000000  00000000  00000000  00000000     *    .           *



Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End
```

Figure 5-15
Table of EBCDIC hex values

Collating Sequence	Character	Hex Value	Collating Sequence	Character	Hex Value
1	Blank	40	49	s	A2
2	¢	4A	50	t	A3
3	.	4B	51	u	A4
4	<	4C	52	v	A5
5	(	4D	53	w	A6
6	+	4E	54	x	A7
7	|	4F	55	y	A8
8	&	50	56	z	A9
9	!	5A	57	{	C0
10	$	5B	58	A	C1
11	*	5C	59	B	C2
12	)	5D	60	C	C3
13	;	5E	61	D	C4
14	¬	5F	62	E	C5
15	- (minus)	60	63	F	C6
16	/	61	64	G	C7
17	¦	6A	65	H	C8
18	,	6B	66	I	C9
19	%	6C	67	}	D0
20	_ (underscore)	6D	68	J	D1
21	>	6E	69	K	D2
22	?	6F	70	L	D3
23	`	79	71	M	D4
24	:	7A	72	N	D5
25	#	7B	73	O	D6
26	@	7C	74	P	D7
27	'	7D	75	Q	D8
28	=	7E	76	R	D9
29	"	7F	77	\	E0
30	a	81	78	S	E2
31	b	82	79	T	E3
32	c	83	80	U	E4
33	d	84	81	V	E5
34	e	85	82	W	E6
35	f	86	83	X	E7
36	g	87	84	Y	E8
37	h	88	85	Z	E9
38	i	89	86	0	F0
39	j	91	87	1	F1
40	k	92	88	2	F2
41	l	93	89	3	F3
42	m	94	90	4	F4
43	n	95	91	5	F5
44	o	96	92	6	F6
45	p	97	93	7	F7
46	q	98	94	8	F8
47	r	99	95	9	F9
48	~	A1			

Figure 5-16

Initialized data sector

```
                    S/36 PATCH DISKETTE UTILITY                                W2

SS@= 0000001    CHR= 010001    Hexadecimal
  Addr    00          04          08          0C
  0000    DB6DB6DB    6DB6DB6D    B6DB6DB6    DB6DB6DB     *__¶__¶__¶__¶__¶_*
  0010    6DB6DB6D    B6DB6DB6    DB6DB6DB    6DB6DB6D     *_¶__¶__¶__¶__¶__*
  0020    B6DB6DB6    DB6DB6DB    6DB6DB6D    B6DB6DB6     *¶__¶__¶__¶__¶__¶*
  0030    DB6DB6DB    6DB6DB6D    B6DB6DB6    DB6DB6DB     *__¶__¶__¶__¶__¶_*
  0040    6DB6DB6D    B6DB6DB6    DB6DB6DB    6DB6DB6D     *_¶__¶__¶__¶__¶__*
  0050    B6DB6DB6    DB6DB6DB    6DB6DB6D    B6DB6DB6     *¶__¶__¶__¶__¶__¶*
  0060    DB6DB6DB    6DB6DB6D    B6DB6DB6    DB6DB6DB     *__¶__¶__¶__¶__¶_*
  0070    6DB6DB6D    B6DB6DB6    DB6DB6DB    6DB6DB6D     *_¶__¶__¶__¶__¶__*
  0080    B6DB6DB6    DB6DB6DB    6DB6DB6D    B6DB6DB6     *¶__¶__¶__¶__¶__¶*
  0090    DB6DB6DB    6DB6DB6D    B6DB6DB6    DB6DB6DB     *__¶__¶__¶__¶__¶_*
  00A0    6DB6DB6D    B6DB6DB6    DB6DB6DB    6DB6DB6D     *_¶__¶__¶__¶__¶__*
  00B0    B6DB6DB6    DB6DB6DB    6DB6DB6D    B6DB6DB6     *¶__¶__¶__¶__¶__¶*
  00C0    DB6DB6DB    6DB6DB6D    B6DB6DB6    DB6DB6DB     *__¶__¶__¶__¶__¶_*
  00D0    6DB6DB6D    B6DB6DB6    DB6DB6DB    6DB6DB6D     *_¶__¶__¶__¶__¶__*
  00E0    B6DB6DB6    DB6DB6DB    6DB6DB6D    B6DB6DB6     *¶__¶__¶__¶__¶__¶*
  00F0    DB6DB6DB    6DB6DB6D    B6DB6DB6    DB6DB6DB     *__¶__¶__¶__¶__¶_*


Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End   Roll keys-Page sector
```

Repairing Damaged Diskettes

by Mel Beckman

When you encounter the dreaded permanent diskette I/O error while restoring a backup file from diskette, SSP forces you to cancel the job — losing the part of the file that was copied successfully. In many cases, though, you might be happy to get as much data as you could, resorting to manual methods to repair damaged records.

An undocumented IBM utility, I1DIAG, lets you locate the bad spots on a diskette and correct them. Although the data stored at the bad location is lost, the utility usually can repair the diskette so that a subsequent restore operation can be completed normally. You carry out the repair process in two steps: the first searches out all bad diskette sectors, and the second rewrites the sectors correctly. You insert the diskette you want to repair in diskette slot S1. Then type:

```
I1DIAG SCAN
```

and press Enter. At the next prompt screen, you press Enter, and I1DIAG scans the diskette, printing a report that notes all bad sectors. After you retrieve the I1DIAG report, type:

```
I1DIAG RECOVER
```

and press Enter. At the prompt screen, you enter the sector address for a bad sector (from the report) and press Enter. I1DIAG attempts to read the sector to recover the data. The utility rewrites recoverable data to diskette several times (you can specify up to 99 times) to ensure that the data "sticks." If the data cannot be recovered, I1DIAG writes a zero sector. You repeat this process for each bad sector on the report.

If the diskette has been damaged physically or the errors are too numerous, I1DIAG may not be able to repair the diskette. But most diskette errors are the result of magnetic, not physical, changes and can be corrected. After correcting all bad sectors and restoring the file to disk, you should check all the records to determine whether any are missing or damaged. Missing records contain binary zeros; damaged records have some fields overwritten by binary zeros.

Retrieving Diskette Available Space and Volume ID

by Simon Kitchen-Dunn

Code on diskette:

Procedure SPACE
RPG program SPACE

Our DP department has a heavy workload relative to staff resources (i.e., both of us are real busy). Therefore, we'd rather not deal with procedures that "bomb" because of an unexpected condition, like having an unattended backup procedure fall over because the diskettes fill up at inopportune moments.

I wrote procedure SPACE (Figure 5-17) and program SPACE (Figure 5-18) to help alleviate this problem. The procedure simply obtains a VTOC listing for the diskette, writes the listing to a file, and calls program SPACE, which reads the VTOC listing file and writes the volume ID and diskette space information to the LDA. The program also writes the number of available bytes, which it computes by multiplying the number of available sectors by the number of bytes per sector. Provided you can compute the amount of diskette space a procedure will require, you can call procedure SPACE from that procedure and compare the amount of space that remains on the diskette to the amount of space needed (Figure 5-19). The procedure then can warn the operator, before the procedure's main task starts, if a new diskette is needed.

Figure 5-17

Procedure SPACE

```
* PARAMETER 1 IS LOCATION, DEFAULT S1
* USE OF LOCAL DATA AREA:-
*                   FROM    TO    DESCRIPTION
*                   ----    --    -----------
*                      1     6    VOLUME IDENTITY
*                      7    10    SECTORS FREE
*                     11    14    BYTES PER SECTOR
*                     15    21    BYTES FREE
// LOAD $LABEL
// PRINTER NAME-$SYSLIST,PRIORITY-0,FORMSNO-VTOC
// RUN
// DISPLAY UNIT-I1,LOCATION-?1'S1'?,LABEL-ALL
// END
// LOAD $UASF
// RUN
```

```
// SPOOL SPOOLID-FVTOC,NAME-VTOCPRT,RETAIN-T,RELCANS-CANCEL
// END
// LOAD SPACE
// FILE NAME-VTOCPRT,RETAIN-S
// RUN
// RETURN
* THESE LINES ARE MERELY TO VERIFY THAT THE PROCEDURE IS WORKING CORRECTLY
// * 'DISKETTE IN POSITION ?1'S1'? VOLUME IDENTITY IS ?L'1,6'?'
// * 'SPACE AVAILABLE ?L'7,4'? SECTORS EACH ?L'11,4'? BYTES, ?L'15,7'? BYTES FREE'
// PAUSE
```

Figure 5-18

Program SPACE

```
*        1         2         3         4         5    .    6         7         8
0001 H     2                                                                 SPACE
0002 FVTOCPRT IP  F 600 150              DISK
0003 IVTOCPRT NS  01
0004 I                                         25  30 ID
0005 I                                         47  500AVAIL
0006 I                                         67  700BPS
0007 I           UDS
0008 I                                          1   6 VOLID
0009 I                                          7  100AVAIL
0010 I                                         11  140BPS
0011 I                                         15  210BFREE
0012 C   01                ADD  1         RECORD  10
0013 C           RECORD    COMP 3                          03
0014 C   03                MOVE ID        VOLID
0015 C           RECORD    COMP 4                          04
0016 C   04      BPS       MULT AVAIL     BFREE
0017 C   04                SETON                    LR
```

Figure 5-19

Sample procedure that calls procedure SPACE

```
// TAG CHECK
SPACE
// IF ?L'1,6'?/CPYINV IF ?L'7,4'?>0000 IF VOLID-'CPYINV,S2' GOTO START
// IFF ?L'7,4'?>0000 ** 'INSERT NEW CPYINV DISKETTE, SLOT 1 FULL!!!!'
INIT CPYINV,,,S1,S2
// GOTO CHECK
// TAG START
// IF ACTIVE-ORDERS * 'ORDER ENTRY ACTIVE'
```

Converting 8-Inch to 5 1/4-Inch Diskettes

by Mark Lazarus, Chuck Lundgren, Jeffrey Pisarczyk, and Bill Roehmer

Q Next week I take delivery of two 5363s, which will replace an overloaded 5362 currently serving (remotely) a main office and a branch office. My problem is that I have no way to copy and load files and programs from the 5362's 8-inch diskettes to the 5363's 5 1/4-inch diskettes. I have talked to both IBM and the third-party vendor who sold me the machines. They insist that changing formats is my responsibility, and they claim not to have the necessary resources to change formats in their offices. In short, I am on my own. Besides a pair of scissors, what do I need to get my 8-inch diskettes down to 5 1/4-inch diskettes?

A Moving your 8-inch diskettes to 5 1/4-inch diskettes may not be as tough as you think it will be, and there is more than one way to do it. First, IBM's Diskette Exchange Utility has software that runs on both the PC/AT and the 5362 that will do the job. This utility requires an IBM

PC/AT with a 5250 emulation board. Please note you can use only an IBM PC/AT — no clones. The program checks to make sure the PC is an authentic PC/AT from IBM.

Second, if you feel comfortable turning your data over to another company, take a look at the media conversion companies in the marketplace. Typically, these companies charge you on a per diskette basis.

Finally, you can buy a separate 8-inch diskette drive and controller that attaches to a PC. With this, you can copy your diskettes directly from an 8-inch drive to a 5 1/4-inch drive without having to go between the PC/AT and 5362. Two or three third-party vendors handle the hardware and software combination you need.

DisplayWrite

CHAPTER

6

Merging Data with DisplayWrite/36 Documents

by Paul Koeller

Whether you choose multicopy merge or column list merge, here are tips and techniques for merging with other DW documents, S/36 files, and Query/36.

Data/text merge functions are some of the most powerful yet least understood functions of DisplayWrite/36 (DW/36). These functions, which let a user merge data into a document created by DisplayWrite, can save hours and, in many cases, eliminate the need to write application programs. This article, which assumes familiarity with DW/36, reviews the basics of creating a shell document and merge processing and provides tips and techniques that experienced users will find helpful.

There are two basic types of merge in DW/36, multicopy merge and column list merge. Multicopy merge provides a mass mailing function. The user creates a shell letter containing both the constant text and the places to insert the variable information (Figure 6-1). In the figure, x's represent variable information merged when the document is printed. Printing the shell document creates multiple copies of the letter, one letter for each record.

Figure 6-1

Multicopy merge example

```
*date

xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx



Dear xxxxxxx,

   As one of the most valued customers in the state of xxxxxxxx, you'll be glad to hear
that we are having a huge sale and you're invited.  The sale will be held on Saturday,
October 10, from 10:00 to 5:00.  Hope to see you there.

Sincerely,

John Smith
```

The other type of merge, column list merge, provides a way to produce a report. The user creates a shell document that defines the format of the report and the data to be merged into the report (Figure 6-2). Again, x's represent variable information merged when the document prints a single report.

Figure 6-2

Column list merge example. This report lists all of our customers. The report is sorted by state, within state by city, and within city alphabetically by customer last name.

STATE	**CITY**	**CUSTOMER NAME**
xxxxxxxxxxxx	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxx	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxx	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxx	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx
		xxxxxxxxxxxxxxxxxxxxxx

After you determine the type of merge you want, you must decide where to get the data that will be merged. You can merge data from three sources: other DW documents, S/36 files, and Query/36. The source you choose depends on several factors: where the data is stored, the amount of data, the complexity of the data, and the complexity of the document that you plan to produce. The source of your data can be specified either on the individual text instructions that control the merge or on page 3 of the print options display.

Merging from a Fill-in Document

One source for your data is another DW document. This source is the easiest to tap, but it is also the most limited. You start by creating a fill-in document containing the names of the fields that you wish to merge along with actual data for those fields. Using a fill-in document to store data eliminates the need to learn about files, dictionaries, queries, and other data processing concepts. Your entire merge application — defining the data, entering the data, and printing the letters — can be accomplished without leaving DW/36.

Before you decide to use a fill-in document for storing data, however, you need to understand its limitations. Data stored in a fill-in document can be used only with a multicopy merge shell document. Also, you can neither produce a column list report from data stored in a fill-in document nor sort data into a different order. And there is no efficient way to select a subset of the records to be merged. Finally, you cannot edit numeric fields merged from a fill-in document.

If you do decide to use a fill-in document, you must choose one of two formats. The first format, column format (Figure 6-3), uses the first line of the fill-in document to define the names and lengths of fields. You key an ampersand (&), the name of the field, and then tab to the right to allow enough space for the maximum length of data that you expect for that field. The process is repeated for each field to be merged. Once the fields have been defined, you enter the data in the columns under the field names. Each line in the document represents one record. Column format fill-in documents work best when the total length of all of the fields does not exceed 80 characters. That way you can enter the data without having to "window" the display to the right for the last fields.

Figure 6-3

Column format fill-in document

&NAME	&STR1	&STR2	&CITY	&ST	&AMT
John Smith	123 Main St	PO BOX 456	Rochester	MN	$123.45
Mary Jones	4567 18th Ave		Austin	TX	$98.56
Tom Johnson	654 Willow Lane	RR1 1	Carmel	NY	$6.18

With the second format of a fill-in document, row format (Figure 6-4), you define each field on a separate line and then leave a blank line to indicate the end of the field names. As before, you key an ampersand (&) followed by the name of the field. Then you press the Field Exit key to start a new line. You repeat this process until all the field names have been entered, and then you press Field Exit one more time to leave a blank line. Once the fields have been defined, you enter the data for each field on a separate line and leave a blank line at the end to indicate the end of the record. Each group of lines represents one record of data. Row format fill-in documents are designed for cases where there are more than 80 bytes of data per record or where the length of the fields varies greatly. If you have several fields, you need to be sure you don't forget to enter data for one of the fields, leaving a line blank. DW/36 assumes a blank line means the end of a record.

Figure 6-4
Row format fill-in document

```
&NAME
&STR1
&STR2
&CITY
&ST
&AMT

John Smith
123 Main St
PO Box 456
Rochester
MN
$123.45

Mary Jones
4567 18th Ave

Austin
TX
$98.56
```

Merging from a File

Another source for merge data could be a S/36 file, either an existing file or one you create for a particular merge. When you merge from a file, you first must use IDDU to define the file. Records are merged in the order they were added to the file. For example, if you create a file with 100 records, you can print 100 copies of a multicopy document or produce a column list report with 100 lines in the report. Merging from a file is best when you have a lot of data but you are not concerned with the order of the records.

Merging from a Query

The final source for your merge data is Query/36. This method is by far the most powerful method used to merge data with DW/36. Suppose you want to create a mass mailing to a customer listing and save postage by sorting the letters in zip code sequence. Simply use Query/36 to create a query that specifies the name of the file, and then select to sort on zip code. When the letters are printed by DW/36, they will be printed in zip code order. Or suppose you want to generate a customer report. In the report, you want to sort the records by state, within state by city, and within city by last name. Furthermore, you want to generate report breaks each time you start a new city or state and have subtotals for each of those breaks that show the number of customers in each city and the minimum, maximum, and average balance due within each city and state. And not only that, you want each state to start on a new page and you need column headings on each page. This sounds like a lot of work, but with the help of DW/36 and Query/36, you can produce such a report quite easily.

You can use Query/36 in several ways. Many people don't realize they can get to Query/36 from DW/36. By pressing Command 17 from the DW/36 edit display, you have access to the full function of Query/36 and also to some special functions that were built just for DW/36. You can create or change a query that specifies

- the file to use
- which fields in the file to use
- the sorting of the fields
- which records from the file to select
- subtotals and totals on the selected fields
- column headings
- numerous other functions available in Query/36

and then return to DW/36. Query/36 offers you three options that save keystrokes and time in DW/36. The options appear as options 6, 7, and 8 on the Work With Queries display (Figure 6-5).

Figure 6-5
Query/Text functions

```
                              WORK WITH QUERIES


 Type choices, press Enter.
  ITEM                     CHOICE     POSSIBLE CHOICES
  Option. . . . . . . . .             1=Create   2=Revise   3=Copy   4=Delete
                                      5=View     6=Text data merge
                                      7=Text column list    8=Text multicopy
  Query name. . . . . . .             Name, blank for list
   Library name . . . . . #QUERY      Name, blank for list









  Cmd7=End         Cmd8=Reset
```

With option 6, Text data merge, the query with which you are working is run and the report is displayed on a split-screen in DW/36. You then can copy all or parts of the report into the shell document you are editing. This option is useful when you want to produce a one-time report with the current data and add text or formatting that isn't available with a normal query report.

With option 7, Text column list, Query/36 creates all of the text instructions needed in your DW document to produce a column list shell document. The instructions are displayed on a split-screen in DW/36 (Figure 6-6). You then can copy the instructions into the document you are editing. The text instructions built include the data fields in the order that you specified in the query, the column headings for the selected fields, and even the running heading instructions to make sure the headings print correctly at the top of each page. Using this option can reduce significantly the time it takes to create column list reports.

Figure 6-6

Text column list returned by Query/36

```
COLLIST,TXTPDK P:12            EDIT Instruction        PG:1      LN:7
<2....:....3....:....4....:....5....v....6....:....7....:....8....:....9>...:..
This is a report that lists our customers, the city they live in, and
the current amount that they owe us.


GUESTQRY,#QUERY P:12        QUERY INPUT                 PG:1      LN:1
 F
*brh
     CUSTOMER NAME            CITY             AMOUNT DUE
*erh
     *&NAME                   *&CITY             *&RATE
*crh
```

Finally, with option 8, Text multi-copy, Query/36 creates multicopy data field instructions for each field specified in the query. Again, these text instructions are displayed on the split-screen in DW/36. You then can copy the instructions into the shell document you are editing.

An extremely powerful function of DW/36 and Query/36, dependent column lists, is also available. By combining the two types of merge — multicopy and column list — it is possible to create a multicopy shell document that, for example, sends one letter to each of your customers and within the letter merges a list of that customer's purchases. This is done by creating two queries: the first query controls the multicopy merge and the second controls the column list merge. This second query is the key to this function. In the second query, you specify to select records that have customer name equal to the customer name currently selected by the first query. For example,

```
NAME EQ NAME(QUERY1,#QUERY)
```

Using the dependent column list function adds powerful versatility to the basic merge function.

Creating a Shell Document

After determining which type of merge you want and where to store and how to retrieve the merge data, you are ready to create the shell document. Your shell document specifies the constant text that appears when the document is printed and the placement of variable data that is merged into the shell document.

DW/36 provides several text instructions that define and control the data merged into your document. The most important instruction, the data field instruction, specifies the name of the field to be merged, the type of merge, and the source of the merge data (Figure 6-7). To create a data field instruction in your document, you either 1) press Command 5 (Goto), type .& in the prompt, and press Enter, or 2) press Command 9 (Text Instructions), select option 12 (Data field), and then select option 1 (Print the data from a data field).

Figure 6-7

Data field instruction display

```
SHELL,TXTPDK P:12                EDIT Instruction         PG:1       LN:12
<2...T:...T3...T:...T4...T:...T5...Tv...T6...T:...T7...T:...T8...T:...T9>...:..

Dear   NAME(*PRINT,,M,,)

You owe us  &AMT(*PRINT,,M,,)


                          DATA FIELD (.&)
 This instruction prints the value of a data field from a described data
 file, query, or document.
 Type choices, press Enter.
  ITEM                    CHOICE        POSSIBLE CHOICES
  Data Field name . . . . . NAME
    File/query/document. . . *PRINT      Name, *PRINT, or *NOTE
      Library/folder . . . .             Name if query or document specified
  Letters or list . . . . . 1            1=Multiple letters     2=Column list
  File id . . . . . . . . .              A=E (for duplicate fields)
  Instruction length . . . .             1=255  (Blank to display entire
                                         instruction)


 Cmd3=Go Back        Cmd5=Numeric editing        Cmd6=Character editing
 Cmd7=End            Cmd14=Subdirectory          Cmd16=Delete instruction
```

Two other useful text instructions are the begin and end conditional text instructions shown in Figure 6-8. They are used in a multicopy shell document to specify text or instructions optionally printed in each letter. This is done by comparing one of the fields from the current record of data to a constant value. Some common reasons for using conditional text are to suppress the printing of blank address lines, to print different salutations based on the sex of the recipient, or to select different paragraphs to be included based on the amount of money a customer owes. Conditional text instructions can be nested up to seven levels deep by performing multiple compares.

Figure 6-9, a typical multiple-letter shell document, shows the use of both data field text instructions and the begin and end conditional text instructions. The symbol * represents the text instruction symbol. The data

Figure 6-8

Begin/End conditional text display

```
SHELL,TXTPOK P:12                 EDIT Instruction        PG:1      LN:16
<2...T:...T3...T:...T4...T:...T5...Tv...T6...T:...T7...T:...T8...T:...T9>...:..

*bct(AMT,*PRINT,,GT,100,,)Because you owe us more than $100.00, we are
adding a $10.00 service charge to your account.
*ect


        BEGIN and END CONDITIONAL TEXT INSTRUCTIONS (.bct / .ect)
 The Begin and End Conditional Text instructions mark the beginning and end
 of the text to be printed when the specified condition is true.
 Type choices, press Enter.
  ITEM                          CHOICE          POSSIBLE CHOICES
  Instruction type . . . . . 1                1=Begin (Enter choices below)   2=End
    Field name . . . . . . . AMT              Data field or variable name
      File/query/document. . *PRINT           File/query/document name or *PRINT
        Library/folder . . .                  If query or document specified
    Selection criteria . . . GT               EQ, NE, GT, GE, LT, LE
    Test value . . . . . . . 100
    File id. . . . . . . . .                  A-E
  Instruction length . . . .                  1-255  (Blank to display entire
                                              instruction)


 Cmd3=Go back    Cmd7=End    Cmd14=Subdirectory  Cmd16=Delete instruction
```

Figure 6-9

Multiple-letter shell document

```
*&NAME(*PRINT,,M,,)
*&STR1(*PRINT,,M,,)
*bct(STR2,*PRINT,,NE,' ',,)*&STR2(*PRINT,,M,,)
*ect*&CITY(*PRINT,,M,,)  *&ST(*PRINT,,M,,)

Dear *&NAME(*PRINT,,M,,)

You owe us *&AMT(*PRINT,,M,,)

*bct(AMT,*PRINT,,GT,100,,)Because you owe us more than $100.00, we are
adding a $10.00 service charge to your account.
*ect
```

field instructions (*&) represent places in the letter where variable information will be inserted into the letter. The first begin conditional text instruction determines whether the contents of field STR2 is equal to blanks. If the field is not equal to blanks, the recipient of the letter has a second address line that will be printed. If the field STR2 is equal to blanks, the recipient has only one address line. The begin conditional text instruction is used to ensure that a blank line is NOT printed in the address.

The second begin conditional text instruction determines whether the value of field AMT is greater than 100. If the value of the field is greater than 100, a sentence is printed telling the customer that a $10 service charge was added to his or her account. If the value is less than or equal to 100, that sentence is not printed.

The data field heading instruction (.dfh) lets you merge up to three lines of heading text for a field. The lines of heading can be defined either in IDDU when you create the data dictionary or in Query/36 when you define the query. These headings are especially useful on a column list shell document. With the data field heading instructions inside running heading instructions (Figure 6-10), DW/36 prints the column headings at the top of each page regardless of the number of pages the column list produces. For example, you might define a column list shell document that produces a list of new customers. Some months you have 10 new customers, and some months you have 1,000. By using running heading instructions to define your headings, DW/36 makes sure that every page gets the correct headings regardless of how many new customers you have this month. In Figure 6-10, the top half of the display shows the text instructions, and the bottom half shows what the printed document will look like.

Figure 6-10

Data field headings with running headings

```
FIGURE10.TXTPDK P:12            EDIT Instruction        PG:1      LN:7
<2....:....3....:....4....:....T....v....6....:....7....:....8....:....9>...:..
This document demonstrates the use of the data field heading text
instruction and running heading text instructions to print column
headings over columnar data that may span several pages.
*brh
*dfh(NAME,*PRINT,,C,)         *dfh(RATE,*PRINT,,C,)
*erh
*&NAME(*PRINT,,M,,)           *&RATE(*PRINT,,M,,)
*crh

FIGURE10.TXTPDKP:12        RESOLVED OUTPUT              PG:1      LN:1

This document demonstrates the use of the data field heading text
instruction and running heading text instructions to print column
headings over columnar data that may span several pages.
 F
THE NAME
OF THE                        CURRENT
PERSON                        AMOUNT DUE
 R
APRIL                         390.00
```

Several other features have been built into DW/36 to make creating shell documents easier. Once you have created one data field instruction in your shell document, DW/36 remembers the source of the data and multiple letter or column list options that you specified on the previous data field instruction. Therefore, you can create additional fields simply by typing a period followed by an ampersand and the name of the field. Another helpful hint is to specify *PRINT for the file/query/document prompt on your data field instructions. This tells DW/36 that, rather than specifying your merge source now, you will provide it when you submit the print request. When you submit the print request, specify the merge source on page 3 of the print options. Using *PRINT is especially useful when you want to merge a shell document with different queries.

After you create your shell document and your data, you are ready to see the results of your work. Using the view print (Command 19) function of DW/36, you can see what your output looks like before you print hundreds of copies of a multicopy letter. When you use view print, DW/36 builds only the first multicopy letter and displays it on a split-screen. An error page is appended to the end of the document. Errors in your shell document, such as a misspelled field name, result in an error message. You can correct those errors before you submit the final printing request.

There are several things to consider when you print a multicopy shell that will produce a large number of letters. Before the Office Enhancements Feature of DW/36, when you printed a multicopy document, DW/36 produced one large work document in a work folder named #TEXTWRK. This single work document could become very large and degrade performance. Furthermore, all of the letters were merged before any of the letters were spooled for printing. With the Office Enhancement Feature, however, 10 multicopy letters are now built in #TEXTWRK and then spooled for printing. You can take advantage of this feature by starting to print those first 10 letters while DW/36 builds the next 10. To do this, use the change option of the DW/36 print queue and change the "Defer printing until complete" prompt to No. This tells the system that the spool file can start printing even though DW/36 is still adding pages to the spool file.

Using the TEXTDOC MERGE Function

Another function of DW/36 that many people may not know about or understand is the TEXTDOC MERGE procedure. Procedure TEXTDOC MERGE lets you merge a shell document while resolving the data field, data field heading, begin/end conditional text, and include (*inc) text instructions. All other text instructions and controls, such as tabs, are left in a form that can be edited. This function is useful if you need to merge the data and the includes into your shell but still do additional editing on the document before you print it. For example, when creating a customer proposal, you may want to merge standard information such as the customer's name and address and several standard paragraphs based on the product in which the customer is interested. Besides the standard information, however, you want to personalize the proposal by adding some text. You could use the TEXTDOC MERGE procedure to merge in the standard information and then edit the resulting document to add the personal messages.

Moreover, with Release 5.1 of DW/36, TEXTDOC Merge is capable of two additional functions. The first is the capability to produce a report that has multiple lines of text for each record merged. For example, in a listing of all of your customers, on the first line you may want the customer's name and address, on the second line his or her account number, and on the next several lines a description of the customer's needs.

You begin by defining a shell document with multicopy data fields (Figure 6-11). In the shell document, begin and end keep-text instructions ensure that none of the records is split across a page boundary when the records are merged. Also, the data field for the description field (DESC) must be preceded by a required tab to indent the text for the description field over to the tab stop in column 55 when it is line-adjusted on subsequent lines.

Figure 6-11

Multiple-line report shell document

```
FIGURE11,TXTPDK P:12              EDIT Instruction          PG:1       LN:7
<2....:....3....:....4....:....v....T....6....:....7....:....>....:....9....:..
*bk
Name/address:    *&NAME(*PRINT,,M,,) *&ADDR(*PRINT,,M,,)
Account number: *&
Description of current needs:       *&DESC(*PRINT,,M,,)*ek
```

Figure 6-12

Multiple-line report output

```
Name/address:    John Smith   123 Main St
Account number:  12345
Description of current needs: Wants more information on using
                              column list merge.

Name/address:    Mary Jones   4567 18th Ave
Account number:  78901
Description of current needs: Wants to understand TEXTDOC
                              MERGE.

Name/address:    Tom Johnson  654 Willow Lane RR1
Account number:  23456
Description of current needs: Wants to use dependent column list
                              merge to create a report.
```

After you have created the shell, use the TEXTDOC MERGE procedure and specify the OPTIONS keyword to display the Merge Options display. On the Merge Options display, specify yes for the "Multiple line report" prompt and specify 3 (Adjust page and line endings) for the "Adjust/paginate options." DW/36 will produce a report for you similar to the one shown in Figure 6-12.

The second new function in TEXTDOC MERGE lets you create an include (*inc) instruction by merging in a data field that contains text specifying the include instruction. To use this function, you must define a data field that contains the exact characters for the include instruction. Enter this as if you were typing the instruction directly on the edit display:

```
.inc(DOCNAME,FLDNAME,n n,)
```

where DOCNAME is the name of the document to be included, FLDNAME is the name of the folder that contains the document, and *n n* is a list of pages to be included optionally.

You might use this function to create an application that builds a sales proposal letter. Suppose your company sells 10 different products. You write an application program your sales people can use to enter a prospective customer's name and address and also to check the products in which the customer is interested. The application program creates a record in a file that contains the customer name, address, and an include instruction that selects pages from the document containing descriptions of each of the 10 products. Then procedure TEXTDOC MERGE merges the three fields into a shell letter; the customer's name and address are merged into the letter, and an include instruction selects the correct pages of product information to be included when the document is printed.

Advanced Tips and Techniques

Now that you are aware of the merge functions available in DW/36, I'll share some tricks that I've found helpful in producing merge documents. In many cases, adjusting the line endings is useful. Specifying yes on the "Adjust line endings" prompt on page 2 of the print options changes the formatting of text merged into a shell document. When merging a long character field into a multicopy shell, for example, this specification causes text that would not normally fit between the margins to wrap around into paragraphs. And by changing your tab stops and using required tabs, you can create indented paragraphs of merged text.

Another technique — ending a line of column list data fields with a carriage return (not a "required carriage return") and then adjusting the line endings — causes the data field to be repeated several times across the line (Figure 6-13). Specify a single column list data field instruction, followed by a tab and a carrier return. Beforehand, set up the document margins and tabs to let three names be merged onto each line. When the document is printed, the "Adjust line endings" prompt on page 2 of the print options is set to yes. The bottom half of the display shows how the data looks when it is merged into the document.

When creating data field instructions, it is helpful to use the display length prompt on the instruction displays to reserve enough space on the edit display for the longest string of data that will be merged. For example, if you're merging a name field and the longest name is 20 characters, set the data field instruction length to 20 so you can see that space reserved on the edit display. This is especially useful if you are using tabs and there is other text on the line after the data field instruction.

Figure 6-13
Adjusting line endings on a column list

```
FIGURE13,TXTPDK P:12             EDIT Instruction         PG:1      LN:7
<....:....3....:....4....:..T.5....:...v6....:....7....T....8....:....9....:.>.
 &NAME(MULTLINE,#QUERY,C,,)



FIGURE13,TXTPDK P:12      RESOLVED OUTPUT                  PG:1      LN:1

John Smith                Mary Jones               Tom Johnson
```

One more technique when using data field instructions helps you specify numeric editing or character editing for the merged data. Numeric editing cannot be used when you are merging from a fill-in document. To specify numeric editing for a data field, press Command 5 from the data field instruction display. This lets you specify numeric editing options such as

- the decimal point character
- the thousands separator character
- how negative numbers are printed
- how leading zeros are handled (e.g., float currency sign)
- if a value of zero is to be printed

To specify character editing for a data field, press Command 6 from the data field instruction display. This facility, which lets you specify options that change the capitalization of character fields, is useful when the data in your file is stored in all capital letters.

Finally, when working with begin and end conditional text, the placement of the instructions causes the most problems. If you want to use conditional text to remove a blank line completely, you need to ensure that the entire line is within the conditional text. This is accomplished most easily by putting the begin conditional text instruction as the very first thing on a line and the end conditional text instruction as the first thing on the next line (Figure 6-9). If you have problems with extra spaces when printing conditional text, look closely at the location of your instructions, and you should be able to see how to get rid of them. Also, you can use the "Adjust line endings" option on page 2 of the print options to adjust the text if you are using conditional text to merge varying length text into the middle of a paragraph.

In summary, DW/36 provides numerous merge functions. So many, in fact, that it can be overpowering. You need to try the functions. Start small and work your way up to advanced merge applications. Experiment and see what happens. You will surprise yourself at what you can accomplish with just a little work.

Merging Printed Output with DisplayWrite/36 Documents

by Paul Podlipny

Code on diskette:
RPG program CASDWM
Procedure CASDWM
Screen format member CASDWMFM

You can improve your on-line documentation using DW/36 to incorporate copies of actual print-keys and sample reports.

Since the advent of DisplayWrite/36 (DW/36) on the S/36, many companies have used it to standardize all their system and application documentation. It makes sense to write documentation on the same system where you are developing applications and performing operations. A major advantage of using DW/36 to create your documentation is the ability to use it as on-line help through help labels embedded in the text that identify help text associated with a particular area of a screen and through text documents stored in folders.

As part of our documentation, our company uses copies of actual application printouts, such as print-keys and sample reports. The simplest way to copy printouts is to extract the data from the spool file using COPYPRT, copy the file into a library member using $MAINT, and use the GET function to merge the data into a DW/36 document. With this method, however, it is impossible to control adequately the formatting of the resulting data, such as page breaks and line spacing. Our solution, utility CASDWM, offers an elegant method of dropping all extraneous data (e.g., spool header and spacing/skipping controls), handling multiple reports, and maintaining the original page integrity of the printouts. This technique makes it easy to incorporate sample reports and print-keys into your on-line documentation and to update the documentation when you change the layouts of your reports.

The five components of the CASDWM utility (see Figure 6-14a for the prompt screen and 6-14b for screen format member CASDWMFM) include procedure CASDWM (Figure 6-15), which controls the overall function of the utility; program CASDWM (Figure 6-16), which processes and reformats the sample reports from the spool file; Interactive Data Definition Utility (IDDU) file definitions (Figures 6-17, 6-18, and 6-19); two queries used by the DW/36 merge function in merging the file into a "shell" DW/36 document (Figures 6-20 and 6-21); and the shell DW/36 document that indicates how the data should be merged and formatted for maintaining page integrity (Figures 6-22 and 6-23a through 6-23g). After these five components have done their work, you use the DW/36 GET function (Command key 14) to include the document that contains the merged data in your application documentation.

Figure 6-14a

Merge prompt screen

```
          **  CASDWM UTILITY  **

     Copy all SPOOL entries with a specified
     name into a DW/36 document

          SPOOL forms name          ****

          Document name    .        ************

          Document folder           ********
```

Procedure CASDWM

Procedure CASDWM begins with the deletion of work files CASDWM and CASDWM?WS?, if they already are on disk, and then prompts for the required parameters via the prompt screen shown in Figure 6-14a. The parameters are the forms name of the group of spool entries you want to merge into your shell document, the name of this new document to be created, and the folder where the document should be stored. The procedure inserts an F in front of the forms-name parameter. We normally put the entries into the spool file with a unique forms name, such as our initials, set up by using the PRINT procedure. To make certain that entries will not be printed before we run procedure CASDWM, we use the PRINT procedure to direct printouts to a printer configured on the system but not attached to a physical device. Next, the procedure runs the $UASF program (the COPYPRT utility), which extracts all spool entries with the specified forms name and writes them to work file CASDWM?WS?. The utility then cancels all the copied spool entries from the spool file.

Program CASDWM

Program CASDWM (Figure 6-16) is the merge program that processes the output from the COPYPRT utility (i.e., file CASDWM?WS?) and copies it into work file CASDWM. The program assigns a report number to each copied spool file entry and writes into the CASDWM output file this report number on every header and detail print line. Next, the program processes all print control information from the input spool file to convert all spacing information into the same number of separate blank lines as the original report. This step also converts all "skip to new line" control characters into blank lines in the CASDWM output file, giving the DW/36 document exactly the same page layouts as the original report. Last, at the end of each report page, the merge program inserts the .pa page instruction that will convert to a DW/36 page instruction when merged into DW/36.

At this point, the program has extracted the data from each original entry in the spool file and written it to a work file. Each entry now has a report number, a header record, a number of detail records, and various blank detail records that have been added to the work file to represent the skipping and spacing control characters found in the original SPOOL file entries.

Defining File CASDWM

You define the file created by program CASDWM to the system by using IDDU. The IDDU header record definition listing (Figure 6-17) shows how to define fields PRC, PRPTNO, and PTTL in a format named CASDWMH. The header record of each report included in the work file contains an H in position 1, which is defined as the record ID for the format. PRPTNO is the two-digit sequential report number assigned by the program to each header record for each report, and PTTL is the title of each report (i.e., the procedure name from the spool file header record).

The IDDU detail print record definition listing (Figure 6-18) shows how to define fields PRC, PRPTNO, and PTXT1, PTXT2, and PTXT3 in a format named CASDWMP. The detail records of each report included in the work file contain a P in position 1, which is defined as the record ID for the format. PRPTNO is the two-digit sequential report number assigned by the program for each report, and PTXT1, PTXT2, and PTXT3 are fields that define the data in each print line of the report. In IDDU, a field may be only 60 characters long, thereby requiring three field definitions to define the full print line.

Figure 6-19 shows the complete file definition listing of file CASDWM composed of the two formats defined in Figures 6-17 and 6-18.

Executing the Merge

Now that you've created file CASDWM and completed the IDDU definitions, you need two queries and a shell DW/36 document to merge the data. The first query, CASDWMQH (Figure 6-20), processes all the H records in file CASDWM. It extracts the H records and writes each of the fields to a 19-byte sequential file, #QRYOUT, in the format defined earlier by IDDU. The second query, CASDWMQP (Figure 6-21), processes the print detail (P) records from the file created by program CASDWM as well as from file #QRYOUT created by the first query. This query contains a little "trick," called a dependent query, and cannot be run outside of DW/36. A dependent query is one that uses a dependent value in the VALUE column of the DW/36 SELECT RECORDS display and can be used only to merge data into a column list in a DW/36 document. It is a reference to another query. In this case, the shaded portion of Figure 6-21 defines the records from file CASDWM that should be selected based on each report number sequentially from file #QRYOUT. This step provides each report with begin format DW/36 instructions before the data, and page end and reset format DW/36 instructions after the data. Both queries are called by the shell DW/36 document (Figure 6-22), merging the data to produce the desired result.

The shell DW/36 document into which all print records associated with each report are inserted consists of DW/36 format instructions only. Figures 6-23a through 6-23c show the prompt screens used to generate the first instruction in the shell document. This instruction F tells DW/36 how to arrange the document and how to print it. Figures 6-23d through 6-23g show the data field prompt screens that instruct DW/36 how to place the data fields (PTTL and PTXT1, PTXT2, and PTXT3) from file CASDWM into the shell document and which query programs to execute to retrieve the data.

These steps create a merged document that contains the original spool reports, each beginning on a new page, in the folder specified on the prompt screen. If the document name already exists in the specified folder, an error message is issued, and the procedure does not replace an existing document. Procedure CASDWM ends with the deletion of file CASDWM and the #QRYOUT files.

Because utility CASDWM always produces a new document, you must use the DW/36 GET function (Command key 14) to include either the entire document or relevant portions in your application documentation. When you execute this step, be careful to copy the formats in front of each page, or the merged data may be adjusted incorrectly in the target document.

Customizing Utility CASDWM

The way the merge function is currently defined, all merged reports are printed on 11-by-8.5 inch paper at 15 characters per inch (cpi), no matter what the original reports' width. However, you may want print-key copies and other reports to print on 8.5-by-11 inch paper at 10 cpi after being merged into DW/36.

One way to do so would be to define on the prompt screen a page format prompt to ask the user which of the two page formats to use. Based on the response, the controlling procedure could direct the processing down one of two paths. The 11-by-8.5 path is described in this article. The 8.5-by-11 path would cause the extracted spool data to be processed by a different program that would truncate the data and write it into a file 85 characters in length. You would have to develop a second set of IDDU definitions and query programs, and you would need a second shell DW/36 document to accommodate the new file and printing requirements.

Despite the initial setup work, the result — improved documentation that includes print-keys and samples of actual application reports appropriately formatted — is certainly worth the effort. The DW/36 merge function achieves this goal easily and effectively.

Figure 6-14b

Screen format member CASDWMFM

```
*... .  1 . .    2 .       3 .. . 4   . .. 5 ..   . 6 .    . 7         8
     SINPUT1
     D          39 415Y               Y                CCopy all SPOOL entries X
     Dwith a specified
     D          26 515Y               Y                Cname into a DW/36 documX
     Dent
     D          26 720Y                                CSPOOL forms name     ..X
     D...
     DSPOOLID    4 747Y  Y                      Y
     D          26 920Y                                CDocument name ..  ... .X
     D  .
     DDOCNAME   12 947Y  Y                      Y
     D          261120Y                       .        CDocument folder        X
     D...
     DDOCFLDR    81147Y  Y                      Y
```

Figure 6-15

Prodecure CASDWM

```
* DisplayWrite Merge Procedure
*    CASDWM form,document,folder
*
// INFOMSG NO
*
// LOAD $DELET
// RUN
// IF DATAF1-CASDWM?WS? REMOVE LABEL-CASDWM?WS?,UNIT-F1
// IF DATAF1-CASDWM     REMOVE LABEL-CASDWM,UNIT-F1
// END
*
// PROMPT MEMBER-CASDWMFM,FORMAT-INPUT1,START-1,LENGTH-'4,12,8'
*
// LOAD $UASF
// RUN
// SPOOL SPOOLID-F?1?,NAME-CASDWM?WS?,RELCANS-CANCEL.
// END
*
// LOAD
// FILE NAME-CASDWS,LABEL-CASDWM?WS?,RETAIN-S,DBLOCK-40
// FILE NAME-CASDWP,LABEL-CASDWM,DBLOCK-40,RECORDS-?F'A,CASDWM?WS?'?,EXTEND-500
// RUN
*
IDDULINK LINK,CASDWM,CASDCT,
*
TEXTDOC MERGE,CASDWM,CASTXT,?2?,?3?,NOREPLACE,NOOPTIONS
*
// LOAD $DELET
// RUN
// REMOVE LABEL-CASDWM,UNIT-F1
// REMOVE LABEL-#QRYOUT,UNIT-F1
// END
*
// RETURN
*
```

Figure 6-16

Merge program CASDWM

```
*      1     . 2       3       4   .   5       6 .     7   .   8
00001H
00002F*************************************************************
00003F*
00004F*  Program Title
00005F*    CAS DisplayWrite Merge Processor
00006F*
00007F*  Description
00008F*    Process Spool File into a Merge Print File
00009F*
00010F*  Halts, Switches and Command Keys
00011F*    None
00012F*
00013F*  Written for CAS by
00014F*    CS / January 1, 1988
00015F*
00018F*
00019F*************************************************************
00020FCASDWS  IP  F 150 150           DISK
00021FCASDWP  O   F 150 150           DISK
00022F*************************************************************
00023I*************************************************************
00024ICASDWS  NS  01   1 CH
00025I                                      12  19 HPRC
00026I                                      42  49 HPRT
00027I*
00028I        NS  02   1NCH
00029I                                    B   1   20DPAG
00030I                                    B   3   40DLIN
00031I                                      11 142 DTXT
00032I*
00033I*************************************************************
00034C*************************************************************
00035C   01               DO                               DO HEADER CALCS
00036C                    MOVE 'H'       PRC     1         REPORT HEADER
00037C          PRPTNO    ADD  1         PRPTNO  20        INCREMENT REPOR
00038C                    MOVELHPRC      PTTL    16        REPORT TITLE
00039C                    MOVE HPRT      PTTL              REPORT TITLE
00040C                    EXCPTRPTHDR                      WRITE REPORT HE
00060C                    Z-ADDO1        WPAG    40        SET NEW PAGE #
00061C                    Z-ADD1         WLIN              SET NEW LINE #
00041C                    END                              END DO HEADER
00042C*
00043C   02               DO                               DO PRINT LINE C
00044C                    MOVE 'P'       PRC               PRINT RECORD
00045C          WPAG      IFNE DPAG                        IF NEW PAGE
00046C                    EXSR NEWPAG                      START NEW PAGE
00047C                    END                              END IF WPAG
00048C          WLIN      IFLT DLIN                        IF NEW LINE
00049C                    EXSR NEWLIN                      SPACE TO NEW LI
00050C                    END                              END IF WLIN
00051C                    MOVELDTXT      PTXT  132         WRITE REPORT LI
00052C                    EXCPTRPTLIN                      WRITE REPORT LI
00053C          DLIN      ADD  1         WLIN    40        NEXT LINE
00054C                    END                              END DO PRINT LI
00055C*
00056C**-------------------------------**
00057C          NEWPAG    BEGSR
00058C**-------------------------------**
00059C                    EXCPTPAGEND                      WRITE NEW PAGE
00060C                    Z-ADDDPAG      WPAG    40        SET NEW PAGE #
00061C                    Z-ADD1         WLIN              SET NEW LINE #
00062C                    ENDSR                            END NEW PAGE
00063C*
00064C**-------------------------------**
00065C          NEWLIN    BEGSR
00066C**-------------------------------**
00067C          WLIN      DO   DLIN      WLIN              DO UNTIL WLIN-D
00068C                    EXCPTBLKLIN                      WRITE BLANK LIN
00069C                    END                              END DO WLIN
00070C                    ENDSR                            END NEW LINE
00071C*
00072C*************************************************************
00073O*************************************************************
00074OCASDWP  E                RPTHDR
00075O                         PRC        1
```

```
000760                        PRPTNO     3
000770                        PTTL      19
000780*
000790       E                RPTLIN
000800                        PRC        1
000810                        PRPTNO     3
000820                        PTXT     135
000830*
000840       E                PAGEND
000850                        PRC        1
000860                        PRPTNO     3
000870                                   6 '.pa'
000880*
000890       E                BLKLIN
000900                        PRC        1
000910                        PRPTNO     3
000920*
000930**************************************************************
```

Figure 6-17
IDDU header record definition

```
                     FORMAT DEFINITION LISTING

Definition name———— CASDWMH        Revision date—— 12/30/87
Data dictionary———— CASDCT         Revised by——— CS

Input record length——— 150           Creation date—— 12/30/87
Output record length——— 150           Created by——— CS
Short comment———— CAS DisplayWrite Merge Report Header

                          FIELD LIST

FIELD     BEGIN    LENGTH  DATA    SHORT COMMENT

PRC           1        1   CHAR    Record Code
PRPTNO        2       2,0  ZONE    Report Number
PTTL          4       16   CHAR    Report Title
*            20      131

                              RECORD ID CODES

   FIELD      POS    TEST   VALUE

 PRC                 EQ       H
```

Figure 6-18
IDDU detail print record definition

```
                     FORMAT DEFINITION LISTING

Definition name———— CASDWMP        Revision date—— 12/30/87
Data dictionary———— CASDCT         Revised by——— CS

Input record length——— 150           Creation date—— 12/30/87
Output record length——— 150           Created by——— CS
Short comment———— CAS DisplayWrite Merge Print Record

                                          FIELD LIST

FIELD     BEGIN    LENGTH  DATA    SHORT COMMENT

PRC           1        1   CHAR    Record Code
PRPTNO        2       2,0  ZONE    Report Number
PTXT1         4       60   CHAR    Report Text 1
PTXT2        64       60   CHAR    Report Text 2
```

```
PTXT3       124     12    CHAR    Report Text 3
*           136     15

                            RECORD ID CODES

    FIELD      POS    TEST   VALUE

    PRC               EQ       P
```

Figure 6-19

IDDU CASDWM file definition

```
                        FILE DEFINITION LISTING

Definition name——— CASDWM               Revision date—— 12/30/87
Data dictionary——— CASDCT               Revised by——— CS

File type———— DISK                   Creation date—— 12/30/87
Max record length——— 150                 Created by——— CS
Short comment——— CAS DisplayWrite Merge File
                                        RECORD FORMAT LIST

  RECORD        INPUT      OUTPUT
  FORMAT       LENGTH      LENGTH    SHORT COMMENT

  CASDWMH         150         150    CAS DisplayWrite Merge Report Header
  CASDWMP         150         150    CAS DisplayWrite Merge Print Record
```

Figure 6-20

Query on header record

```
5727QU1QU   RO5MOO   IBM SYSTEM/36 QUERY    09/12/88    15 01 49   PAGE 1

   QUERY NAME ———————— CASDWMQH
   LIBRARY ————————— CASLIB
   QUERY DESCRIPTION ———— CAS DisplayWrite Merge Report Header Query

   FILE NAME ————————
   DICTIONARY ———————— CASDCT
   FILE DEFINITION NAME ———
   RECORD FORMAT ——————— CASDWMH

   COLLATING SEQUENCE ——— EBCDIC

           SELECT RECORDS

   AND/     FIELD
   OR       NAME      TEST     VALUE

** No record selection tests, so all records selected
                             SELECT FIELDS

   FIELD     SORT         ASCENDING/
   NAME      PRIORITY     DESCENDING        COMMENTS

   PRC                                      Record Code
   PRPTNO                                   Report Number
   PTTL                                     Report Title
                            FORMAT AND SUMMARIZE COLUMNS

   SUMMARY FUNCTIONS   1-TOTAL 2-AVERAGE  3-MINIMUM  4-MAXIMUM  5-COUNT
                                                                      OVER-
RIDE
   FIELD     SUMMARY        COLUMN      COLUMN                 DEC        DEC
   NAME      FUNCTIONS      SPACING     HEADING            LEN  POS  LEN  POS

   PRC                      0           PRC                1
```

```
PRPTNO                  2          PRPTNO                2     0
PTTL                    2          PTTL                  16

                        SELECT OUTPUT DEVICE

OUTPUT DEVICE -------- DISK
TYPE OF OUTPUT ------- DETAIL

DISK FILE DETAILS

FILE NAME ----------- #QRYOUT
DATA IN FILE -------- NEW
PRINT DEFINITION ----- NO

                DISK OUTPUT FILE RECORD FORMAT

OUTPUT RECORD LENGTH ----   19

                     FIELD LIST

FIELD     BEGIN   LENGTH  DEC POS   DATA     SHORT COMMENT

PRC       1       1                 CHAR     Record Code
PRPTNO    2       2       0         ZONE     Report Number
PTTL      4       16                CHAR     Report Title
```

Figure 6-21

Query on detail print record

```
QUERY NAME ---------- CASDWMQP
LIBRARY ------------ CASLIB
QUERY DESCRIPTION ------- CAS DisplayWrite Merge Report Line Query

FILE NAME -----------
DICTIONARY ---------- CASDCT
FILE DEFINITION NAME ----
RECORD FORMAT -------- CASDWMP

COLLATING SEQUENCE ------ EBCDIC

         SELECT RECORDS

AND/     FIELD
OR       NAME      TEST    VALUE

         PRPTNO     EQ     PRPTNO(CASDWMQH.CASLIB)

                        SELECT FIELDS

FIELD     SORT        ASCENDING/
NAME      PRIORITY    DESCENDING        COMMENTS

PRC                                     Report Code
PRPTNO                                  Report Number
PTXT1                                   Report Text 1
PTXT2                                   Report Text 2
PTXT3                                   Report Text 3

                        FORMAT AND SUMMARIZE COLUMNS

SUMMARY FUNCTIONS   1-TOTAL 2-AVERAGE  3-MINIMUM  4-MAXIMUM  5-COUNT

                                                                  OVER-
RIDE
FIELD     SUMMARY        COLUMN       COLUMN                   DEC        DEC
NAME      FUNCTIONS      SPACING      HEADING           LEN   POS   LEN   POS

PRC                      0            PRC               1
PRPTNO                   2            PRPTNO            2     0
PTXT1                    2            PTXT1             60
PTXT2                    2            PTXT2             60
PTXT3                    2            PTXT3             12

                        SELECT OUTPUT DEVICE

OUTPUT DEVICE -------- DISK
```

```
TYPE OF OUTPUT -------- DETAIL

                         DISK FILE DETAILS

FILE NAME ------------ #QRYOUT
DATA IN FILE ---------- NEW
PRINT DEFINITION ------- NO

                   DISK OUTPUT FILE RECORD FORMAT

OUTPUT RECORD LENGTH ----- 135

                         FIELD LIST

FIELD     BEGIN   LENGTH  DEC POS   DATA      SHORT COMMENT

PRC          1      1               CHAR      Record Code
PRPTNO       2      2         0     ZONE      Report Number
PTXT1        4     60               CHAR      Report Text 1
PTXT2       64     60               CHAR      Report Text 2
PTXT3      124     12               CHAR      Report Text 3
```

Figure 6-22

DW/36 merge document

```
CASDWM,CASTXT P:15               EDIT Format Change          PG:1        LN:7
 <       . 2  ... .  3..    . ..4  ..·.   5  .    6       . .7      ...8
  F
  *&PTTL

  *&PTXT1*&PTXT2*&PTXT3
  pa
  R
```

Figure 6-23a

Typestyle prompt screen

```
                           TYPESTYLE/COLOR
Format Change                                                Menu bypass  t
Type choices, press Enter.
  ITEM                        CHOICE   POSSIBLE CHOICES
  Typestyle (Pitch)       .   230      1-65     (10),      66-153  (12),
                                       154-200 (PSM),      211-239 (15),
                                       240-249 (5),        250-259 (17 1),
                                       260-279 (8.55)
  Color            . . . .    0        0=Base      1=Blue      2=Red
                                       3=Pink      4=Green     5=Turquoise
                                       6=Yellow    8=Black    16=Brown

Enter=Continue                Cmd3=Go back             Cmd7=End
```

Figure 6-23b

Page layout prompt screen

```
                  PAGE LAYOUT/PAPER OPTIONS (2 of 2)
Format Change                                                Menu bypass  p
Type choices, press Enter
  ITEM                     CHOICE  POSSIBLE CHOICES
  Paper width              11       1-45.5 inches
  Paper length             8.5      1-45.5 inches
  Printing paper source,
    first page             1       1-3=Paper drawer      4=Manual feed
    following pages .      1       5=Envelope feed       6=Continuous feed
  Rotate paper  . .        1       1=Auto   2=0   3=90   4=180   5=270 (degrees)
  Print header on          2       1=All pages           2=Following pages
  Print footer on          2       1=All pages           2=Following pages

Enter=Continue      Cmd3=Go back       Cmd7=End      Roll=Previous options
```

Figure 6-23c

Margins and tabs prompt screen

```
                            MARGINS AND TABS
                                                        1         1         1
1        2         3         4         5         4         5         6
0        0         0         0         0         0         0         0
<     T    T    T    T    T    T    T    T./\/\/ T.      T    T   ·T    T >.3.
 Format Change                                              Menu bypass  m
 CHANGE MARGIN      Move cursor to where you want margin, type:
                      <  (for Left Margin)          >  (for Right Margin)
 CLEAR ALL TABS     Press Cmd4
 CLEAR TAB          Move cursor to the tab setting   Press the space bar

 SET TAB            Move the cursor to where you want the tab setting
                    -For Flush Left Tab, type T  -For Center Tab, type C
                    -For Decimal Tab, type D     -For Flush Right Tab, type R
                    -For Comma Tab, type M       -For Colon Tab, type N

Enter-Continue                 Cmd3-Go back                Cmd4-Clear tabs
Cmd7-End                       Cmd10-Window
```

Figure 6-23d

PTTL data field instructions

```
                           DATA FIELD (.&)
This instruction prints the value of a data field from a described data
file, query, or document
Type choices, press Enter
  ITEM                        CHOICE        POSSIBLE CHOICES
  Data field name             PTTL
    File/query/document       CASDWMQH      Name, *PRINT, or *NOTE
      Library/folder          CASLIB        Name if query or document specified
  Letters or list             1             1-Multiple letters     2-Column list
  File id                                   A-E (for duplicate fields)
  Instruction length            6           1-255  (Blank to display entire
                                            instruction)
Cmd3-Go back        Cmd5-Numeric editing         Cmd6-Character editing
Cmd7-End            Cmd14-Subdirectory           Cmd16-Delete instruction
```

Figure 6-23e

PTXT1 data field instructions

```
                           DATA FIELD (.&)
This instruction prints the value of a data field from a described data
file  query, or document
Type choices, press Enter
  ITEM                        CHOICE        POSSIBLE CHOICES
  Data field name             PTXT1
    File/query/document       CASDWMQP      Name, *PRINT, or *NOTE
      Library/folder          CASLIB        Name if query or document specified
  Letters or list             2             1-Multiple letters     2-Column list
  File id                                   A-E (for duplicate fields)
  Instruction length            7           1-255  (Blank to display entire
                                            instruction)
Cmd3-Go back        Cmd5-Numeric editing         Cmd6-Character editing
Cmd7-End            Cmd14-Subdirectory           Cmd16-Delete instruction
```

Figure 6-23f

PTXT2 data field instructions

```
                         DATA FIELD (.&)
This instruction prints the value of a data field from a described data
file, query, or document.
Type choices, press Enter.
  ITEM                      CHOICE         POSSIBLE CHOICES
  Data field name     . .   PTXT2
    File/query/document  . . CASDWMQP      Name, *PRINT, or *NOTE
       Library/folder . . . . CASLIB       Name if query or document specified
  Letters or list  . . . . . 2             1=Multiple letters    2=Column list
  File id         . . . . .                A-E (for duplicate fields)
  Instruction length . . .     7           1-255  (Blank to display entire
                                           instruction)
Cmd3=Go back        Cmd5=Numeric editing       Cmd8=Character editing
Cmd7=End            Cmd14=Subdirectory         Cmd18=Delete instruction
```

Figure 6-23g

PTXT3 data field instructions

```
                         DATA FIELD ( &)
This instruction prints the value of a data field from a described data
file, query, or document.
Type choices, press Enter.
  ITEM                      CHOICE         POSSIBLE CHOICES
  Data field name    . . . . PTXT3
    File/query/document.   . CASDWMQP      Name, *PRINT, or *NOTE
       Library/folder . . . . CASLIB       Name if query or document specified
  Letters or list  . . . . . 2             1=Multiple letters    2=Column list
  File id  . . .   . . . .                 A-E (for duplicate fields)
  Instruction length . . .     7           1-255  (Blank to display entire
                                           instruction)
Cmd3=Go back        Cmd5=Numeric editing       Cmd8=Character editing
Cmd7=End            Cmd14=Subdirectory         Cmd18=Delete instruction
```

Integrating Application Programs and DisplayWrite/36

by Nancy R. Vogelsang and Tammy A. Zitzmann

You can combine an RPG program and DisplayWrite/36 in a single procedure.

Among the various strengths of the S/36, one key strength is its word processing ability. The S/36 supports DisplayWrite/36 (DW/36), a full-function word processor. While you can use Query/36 to merge your data processing information into your word processing documents, combining DW/36 with RPG for the same purpose may seem questionable. But when we attempted it, we found it to be a match made in heaven.

One of our clients, a public health nursing agency, asked us for help automating several of its federally mandated forms. Because we could create most of these forms using DW/36, we proceeded smoothly on the project. But when we came to one set of forms — form 485 and form 487 — that required a combination of heavy-duty word processing and data processing, we knew DW/36 would need some outside help.

Form 485 (Figure 6-24a) contains basic information about a patient; form 487 (Figure 6-24b) contains additional information that does not fit on form 485. Some of the information needed on these forms is contained in files already resident on the S/36 (shown in Figures 6-24a and 6-24b), some requires the operator to check the correct boxes (shown in Figure 6-24a), and some requires the convenience of word processing, such as word wrap and spell checking (shown in Figures 6-24a and 6-24b).

We could easily use DW/36 to create form 487. Box 7 is constant data and the operator could key box 8 and merge the data for boxes 1 through 6 from existing data files. But creating a shell document for form 485 presented a problem because the transcriber must key in variable-length text (e.g., box 10). Not knowing in advance how long the text will be makes it impossible to hard code the correct number of carrier returns in the shell format to allow for printing within the appropriate box. Also, DW/36 cannot tell the printer to skip to a specific line number.

In designing our solution, we knew we wanted to use the data already contained in the S/36 files to fill in appropriate boxes on the forms, and we knew that capturing and presenting the questions for the other boxes in a sequential order was critical because the operator would be keying from dictaphone tapes created by the nurses. Our first approach was to use an RPG program for form 485; print "SEE ATTACHED FORM 487" in boxes 10, 21, and 22, which require heavy text entry; and use DW/36 to create form 487. We ran into a bureaucratic obstacle, however, in that the federal government requires text in these boxes; only continuing text can be on form 487.

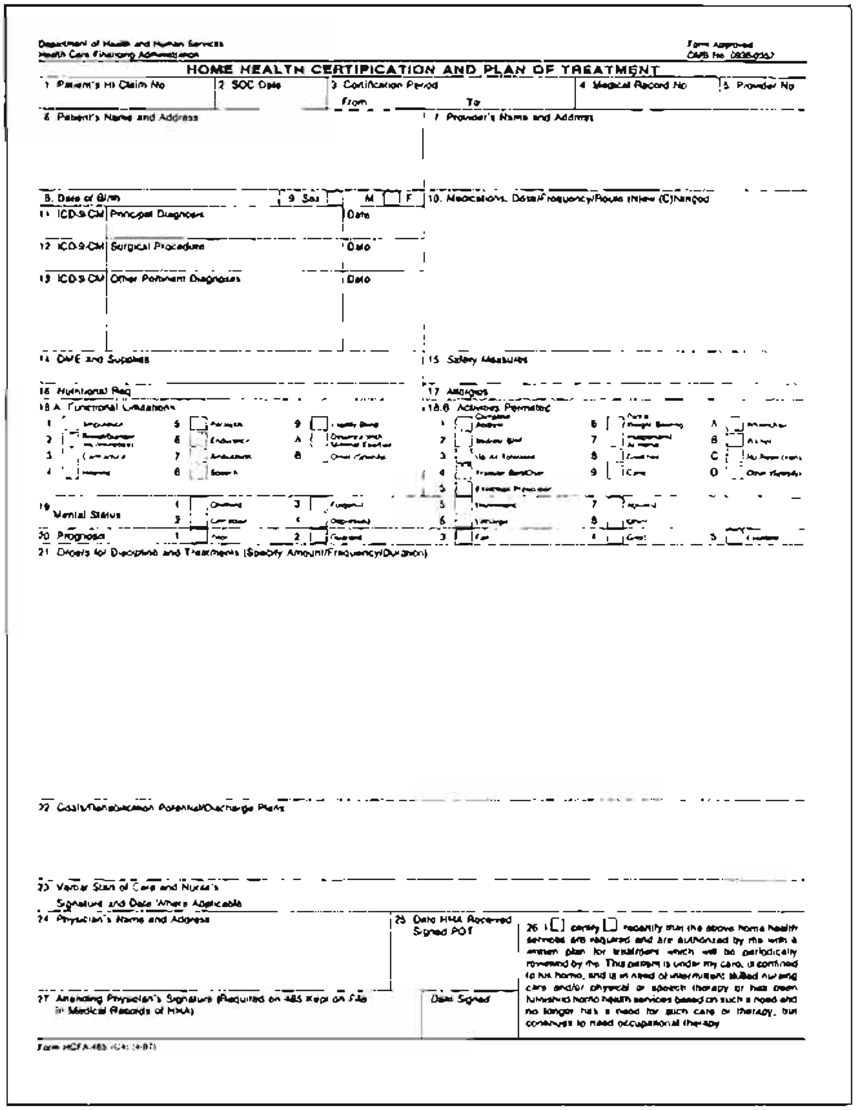
After carefully analyzing the problem, we remained committed to RPG for printing form 485. Using RPG, however, requires working within certain constraints. It means sacrificing word wrap and spell check for the few lines contained in these boxes. It also means that after filling in one of these boxes, the operator must be able to key any additional text on form 487 sequentially without having to search the dictaphone tape. The application must let the operator leave the RPG program, go to DW/36, type the remaining text for any of these boxes, and then return to the RPG program at the point of exit.

The Actual Steps

Let's look now at how we achieved this merger of RPG and DW/36. Procedure PRT485 (Figure 6-25) controls the merger. First, procedure PRT485 calls program PRTBEG (Figure 6-26), which prompts for the patient number to use in creating work files. Two one-record files provide the interface between an RPG program that prints form 485 (in our case, program PRT485) and DW/36. The first file — PATSV, containing all the information entered on form 485 — displays previously entered information when control is returned to the print program. The second file — PATWRK, containing only the information necessary to print the top boxes on form 487 — is merged with the DW/36 document when form 487 is printed. To ensure unique file names, we embed the patient number in the file name. Using Z group files (Z, because there were none on the system), we label the PATWRK file Z.*xxxx* and the PATSV file Z.*xxxx*SV, where *xxxx* is the four-digit patient number.

Figure 6-24a

Form 485



HOME HEALTH CERTIFICATION AND PLAN OF TREATMENT

1 Patient's HI Claim No | 2 SOC Date | 3 Certification Period From To | 4 Medical Record No | 5 Provider No

6 Patient's Name and Address | 7 Provider's Name and Address

8 Date of Birth | 9 Sex M F | 10 Medications: Dose/Frequency/Route (N)ew (C)hanged

11 ICD-9-CM Principal Diagnosis Date

12 ICD-9-CM Surgical Procedure Date

13 ICD-9-CM Other Pertinent Diagnoses Date

14 DME and Supplies | 15 Safety Measures

16 Nutritional Req. | 17 Allergies

18.A. Functional Limitations | 18.B. Activities Permitted

19 Mental Status

20 Prognosis

21 Orders for Discipline and Treatments (Specify Amount/Frequency/Duration)

22 Goals/Rehabilitation Potential/Discharge Plans

23 Verbal Start of Care and Nurse's Signature and Date Where Applicable

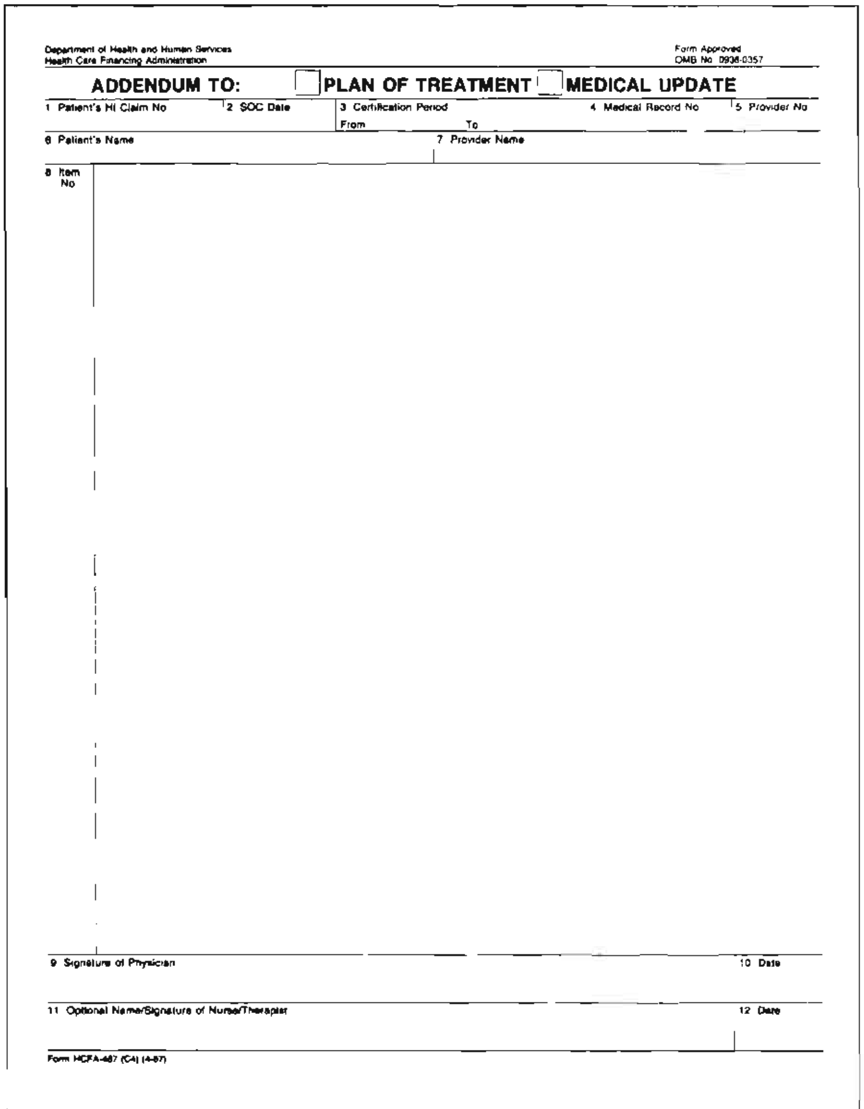
24 Physician's Name and Address | 25 Date HHA Received Signed POT

27 Attending Physician's Signature (Required on 485 Kept on File in Medical Records of HHA) | Date Signed

Program PRTBEG passes a valid patient number back to procedure PRT485 in positions 1 through 4 of the LDA. To avoid the overhead of using a BLDFILE to create the PATSV file, we define the file twice in the print program (i.e., once as an output file, and once as an update/chain file) and condition the use of the correct file on the appropriate external switch. Before loading the print program, the procedure determines whether the operator has created forms for this patient previously. If forms have been created, Z.*xxxx*SV

Figure 6-24b

Form 487



Department of Health and Human Services
Health Care Financing Administration

Form Approved
OMB No. 0938-0357

ADDENDUM TO: ☐ PLAN OF TREATMENT ☐ MEDICAL UPDATE

1 Patient's HI Claim No | 2 SOC Date | 3 Certification Period From To | 4 Medical Record No | 5 Provider No

6 Patient's Name | 7 Provider Name

8 Item No

9 Signature of Physician | 10 Date

11 Optional Name/Signature of Nurse/Therapist | 12 Date

Form HCFA-487 (C4) (4-87)

exists and switch 4 is set on to update it; the print program chains to the PATSV file to display the previously entered information. If forms have not been created, Z.*xxxx*SV does not exist and switch 3 is set on to create the file.

The print program displays the first screen (Figure 6-27a). If the operator happened to key the wrong patient number, he or she now can press Command key 3, which causes switch 1 to be set on. The procedure returns to the PRTBEG program to prompt for another patient number. If the patient number is correct, the operator can begin to key the required information. If the operator needs to key additional information for box 10, he or she presses Command key 5 to interface with DW/36. The program sets on switch 2 and places a 1 in position 5 of the LDA so that on return to the print program, the correct screen will be displayed with the cursor positioned at the correct field.

Control returns to procedure PRT485, which activates the DW/36 interface. When activating the DW/36 interface, the procedure skips the step to LINK the file; we need to do that only before we print the document. If the operator is creating the forms for this patient (i.e., if switch 3 is on), the shell format is copied and a new document is created. We call our new document Zxxxx to make it easy for the operator to know what file name is to be merged with DW/36. Next, procedure PRT485 calls the DW/36 editor to let the operator enter text for form 487 into the just-created document. The information in boxes 1 through 6 will be merged from the Z.xxxx file; box 7 is constant text; box 8 is where the operator starts keying the continued text.

Because DW/36 doesn't allow data fields in header margin text, we inserted the data fields directly in the document. For operator ease of use, the first line of text in the document contains a DW/36 comment instruction. The comment, a form of on-line documentation, informs the operator to press Command N (next stop code) to position the cursor at the appropriate place to begin typing the remaining text for box 10. After completing the text entry, the operator presses Command key 7 and chooses not to print the document at this time. Upon exiting from DW/36, procedure PRT485 sets switches 3 and 4 to indicate the existence of the PATSV file (i.e., switch 3 is set off, switch 4 is set on).

The procedure again loads the print program, which displays the correct screen with the cursor positioned at the next field to be entered. The operator continues entering data. For operator ease of use, when creating the $SFGR screen specifications, we specified controlled field exit on each input field and null fill on each format. The operator fills in information on screens 2 and 3 (Figures 6-27b and 6-27c), interfacing with DW/36 through Command key 5 as necessary. On screen 4 (Figure 6-27d), the operator enters the additional doctors as required; one form will be printed for each doctor entered.

The program sets off switch 2 when exiting the program from screen 4, causing procedure PRT485 to execute an IDDULINK to link the IDDU specifications to the Z.xxxx file. The procedure again calls the DW/36 editor to let the operator key any additional text for form 487. When the operator has completed keying the entire text for form 487, he or she performs a spelling check using the medical supplement to the DW/36 dictionary. Now when the operator presses Command key 7 to end the edit session, he or she selects printing, along with a display of the print options. On page 3 of the print options display (Figure 6-28), the operator defines the name of the file at the File/query/document prompt. The file name, thanks to our naming convention, is the same as the document name, except for the . between the Z and the number. The operator can find the document name in the upper left corner of the display. Once the operator has defined the file name, he or she is finished processing this patient and can return to program PRTBEG to prompt for the next patient number.

The technique we've demonstrated here could be used to combine any RPG program with DW/36. It's one way to merge the power of word processing and data processing.

Figure 6-25

Procedure PRT485

```
* ENTER AND PRINT FORMS 485 and 487
*
*  SWITCH SETTINGS
*  U1 - set on in PRT485 to recycle to prompt for patient number
*  U2 - set on in PRT485 to go to and then return from DW/36
*  U3 - Create Z.xxxxSV file
*  U4 - Update Z.xxxxSV file
*  U8 - set on in PRTBEG to indicate end of procedure
*
*  LDA USAGE
*  Positions 1 - 4 Patient Number
*  Postion 5       Screen to display when returning to PRT485
*
// TAG AGAIN
// LOCAL OFFSET-1,BLANK-5
// SWITCH 00000000
*  Prompt for Patient Number
// LOAD PRTBEG
// FILE NAME-WCHMST,DISP-SHR
// FILE NAME-WCHLST,LABEL-WCHMSTYE,DISP-SHR
// RUN
// IF SWITCH8-1 RETURN
*  Determine status of files Z.xxxx and Z.xxxxSV
// IFF DATAF1-Z.?L'1,4'?SV GOTO CHKSW
// IF ?F'A,Z.?L'1,4'?SV'?>00000000 GOTO CHKSW
// LOAD $DELET
// RUN
// SCRATCH UNIT-F1,LABEL-Z.?L'1,4'?SV
// END
// TAG CHKSW
// IF DATAF1-Z.?L'1,4'?SV SWITCH XX01XXXX
//                    ELSE SWITCH XX10XXXX
// IFF DATAF1-Z.?L'1,4'? GOTO PRT485
// LOAD $DELET
// RUN
// SCRATCH UNIT-F1,LABEL-Z.?L'1,4'?
// END
*  Print form 485
// TAG PRT485
// LOAD PRT485
// FILE NAME-WCHMST,DISP-SHR
// FILE NAME-WCHLST,LABEL-WCHMSTYE,DISP-SHR
// FILE NAME-WCHDOC,DISP-SHR
// FILE NAME-ICDA,DISP-SHR
// FILE NAME-PATWRK,RECORDS-1,LABEL-Z.?L'1,4'?
// FILE NAME-PATSV,RECORDS-1,LABEL-Z.?L'1,4'?SV
// FILE NAME-PATUP,LABEL-Z.?L'1,4'?SV
// PRINTER NAME-PRT485,FORMSNO-485,LINES-66,LPI-6,PRIORITY-0,DEVICE-P2
// RUN
*  Return to prompt for different Patient No
```

```
// IF SWITCH1-1 GOTO AGAIN
* Interface to DW/36
// TAG DW36
// IF SWITCH2-0 IDDULINK LINK,Z,?L'1,4'?,NEWDICT,PATWRK
// IF SWITCH3-1 TEXTDOC COPY,487SHELL,ADDENDUM,Z?L'1,4'?,TXT485
TEXTDOC REVISE,Z?L'1,4'?,TXT485
// SWITCH XX01XXXX
* DW/36 Interface active, return to PRT485
// IF SWITCH2-1 GOTO PRT485
* Prompt for next Patient No.
// GOTO AGAIN
```

Figure 6-26

Program PRTBEG

```
*  . .  1  .    2 .   .  3 ..    4      . 5      . 6      . 7      8
H                                                                 PRTBEG
FPRTBEGFMCP  F      40              WORKSTN
FWCHMST  IC  F 300 300R 4AI      2 DISK
FWCHLST  IC  F 300 300R 4AI      2 DISK
E                      MSG     1   1 40
IPRTBEGFMNS  01    1 C1
I                                       2   50PATNO
I        NS  09
IWCHMST  NS
IWCHLST  NS
ILDA        UDS
I                                       1   40PATNO
C                        SETOF                     10
C   KG                   SETON                     LRU8
C   KG                   GOTO END
C/SPACE
C   09                   GOTO END
C/SPACE
C   01         PATNO     CHAINWCHMST               10
C   01 10      PATNO     CHAINWCHLST               10
C   01N10                SETON                     LR
C/SPACE
C              END       TAG
OPRTBEGFMD         09
O                                    K8 'SCRN1   '
O          D       01 10
O                                    K8 'SCRN1   '
O                  10       MSG,1    40
** MSG
INVALID PATIENT NUMBER
```

Figure 6-27a

Prompt screen 1

```
 1           HOME HEALTH CERTIFICATION AND PLAN OF TREATMENT

Patient Name

10  Medications  Dose/Frequency/Route (N)ew (C)hanged







12  Surgical Procedure

14  DME and Supplies

15  Safety Measures

16  Nutritional Requirements

17  Allergies

CMD 3:OOPS! wrong patient-try again     CMD 5:DW/36     ENTER:Continue
```

Figure 6-27b
Prompt screen 2

```
 2            HOME HEALTH CERTIFICATION AND PLAN OF TREATMENT

Patient Name

 18A Functional Limitations
 0 1 Amputation          0 5 Paralysis             0 9 Legally Blind
 0 2 Bowel/Bladder       0 6 Endurance             0 A Dyspnea
 0 3 Contracture         0 7 Ambulation
 0 4 Hearing             0 8 Speech                0 B Other

 18B Activities Permitted
 0 1 Complete Bedrest    0 6 Partial Weight Bearing 0 A Wheelchair
 0 2 Bedrest BRP         0 7 Independent at Home   0 B Walker
 0 3 Up as Tolerated     0 8 Crutches              0 C No Restrictions
 0 4 Transfer Bed/Chair  0 9 Cane
 0 5 Exercises Prescribed                          0 D Other

 19  Mental Status
     0 1 Oriented  0 3 Forgetful      0 5 Disoriented     0 7 Agitated
     0 2 Comatose  0 4 Depressed      0 6 Lethargic       0 8 Other

 20  Prognosis  0 1 Poor   0 2 Guarded   0 3 Fair   0 4 Good   0 5 Excellent

 CMD 3:Previous Screen                            ENTER:Continue
```

Figure 6-27c
Prompt screen 3

```
3            HOME HEALTH CERTIFICATION AND PLAN OF TREATMENT

Patient Name

21  Orders for Discipline and Treatments








                                                                  More 0
22  Goals/Rehabilitation Potential/Discharge Plans

                                                                  More 0

CMD 3:Previous Screen         CMD 5:DW/36           ENTER:Continue
```

Figure 6-27d

Prompt screen 4

```
4           HOME HEALTH CERTIFICATION AND PLAN OF TREATMENT

Patient Name
_______________________________________
      Doctor                             License
      Address..

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      Doctor·                            License
      Address..

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      Doctor                             License
      Address

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      Doctor                             License·
      Address. .

CMD 3:Previous Screen                        ENTER:Print 485
```

Figure 6-28

D/W/36 print options display–screen 3

```
Z1234.TXT485                PRINT OPTIONS                Page 3 of 3

Type choices, press Enter

ITEM                          CHOICE        POSSIBLE CHOICES
Print revision symbols . .    2             1=Yes    2=No
  Symbols to be printed

Cancel on error  . .          2             1=Yes    2=No
Print error log  .       .    2             1=Yes    2=No
  Forms number for error log                Printer form
  Clear log before printing   1             1=Yes    2=No

File/query/document           Z.1234        File/query/document name
  Library/folder                            If query or document specified

Save printed output           2             1=Yes    2=No
  Document name                             Blank for list of documents
    Folder name        .                    Blank for list of folders

Cmd2=Save in PC file     Cmd3=Go back     Cmd5=Print Queue       Cmd7=End
Cmd9=Change format options                Roll down=Additional print options
Cmd14=Subdirectory:printed output         Cmd15=Subdirectory:query document
```

Assigning #LIBRARY as DisplayWrite/36 Default Library

by Larry N. Forrister

Code on diskette:
Procedure TEXTDOC
Screen format member #0
Message menu member #0##

When users in my shop sign on to DisplayWrite/36, their application library is assigned to their session by default because SSP is unaware they no longer need it. A problem results when a library backup or reorganization requires users to stop work and sign off, causing considerable inconvenience to the DW/36 users.

To alleviate the problem, I wrote procedure TEXTDOC to transfer these users from the application library to #LIBRARY automatically. And through a new menu, users can return to their previous menu and library with a single keystroke.

Procedure TEXTDOC (Figure 6-29) is placed in each affected application library to be invoked in place of IBM's TEXTDOC. After reassigning the user's session to #LIBRARY, my procedure invokes the IBM procedure. When the user is through with DW/36, menu #0 (Figures 6-30a and 6-30b) is displayed so the user can return to the previous menu and library by simply pressing Command 3. Menu load members #0 and #0## (Figure 6-31) must be placed in #LIBRARY when compiled. This technique also works if DW/36 is invoked from a help menu.

Figure 6-29
Procedure TEXTDOC

```
// LIBRARY NAME-O          Change current library.
// MENU #O.#LIBRARY        Change session menu and session library.
// RESET TEXTDOC *ALL      End current procedure and start real job.
```

Figure 6-30a
Menu #0

```
                    Press Cmd3 to return to previous menu

                    1  Return to Displaywrite/36
                    2. Sign OFF the system.


    Cmd3-Previous menu        Cmd5-Main help menu        Home-Sign on menu
```

Figure 6-30b

Screen format member #0

```
*. . ... 1      .. 2 ... ... 3 . . .   4  .     5 .      6   .     7 ..     8
    S*  FREE  FORM MENU
    S#0                   YY       Y                          56C
    DWSID         2 178Y
    DINPUT      12022 3   Y
    DCOMMAND      7 1 2Y        Y                        CCOMMAND
    DINQUIRY      7 169Y                     05Y         CINQUIRY
    DMNUTITL     40 120Y               Y                 C                      X
    D
    DPRMPT1      3821 2Y                                 C                      X
    D
    DPRMPT2      372142Y                     05Y         CCmd1-Resume job
    D
    DFM0001      381214Y                                 CPress Cmd3 to return to X
    D previous menu.
    DFM0002      291414Y                                 C1  Return to Displaywri X
    Dte/36.
    DFM0003      231514Y                                 C2  Sign OFF the system
    DFM0004      1819 2Y                                 CCmd3-Previous menu
    DFM0005      191927Y                                 CCmd5-Main help menu
    DFM0006      171953Y                                 CHome-Sign on menu
```

Figure 6-31

Menu command source member #0##

```
#0##.2
0001 TEXTDOC
0002 OFF
```

Accessing PC DisplayWrite/3 Documents from DisplayWrite/36

answered by Georgia Agallianos

Q I just started using DisplayWrite/36 (DW/36) on my S/36; I also have 5250 emulation on a PC/XT, which is loaded with DisplayWrite/3 (DW/3). One of the selling points of using DW/3 on a PC over using DW/36 on the S/36 is to cut down on I/O on the S/36. Can I upload DW/3 documents to DW/36 on the S/36?

A You can use DW/3 on the PC and simply transfer the documents to the S/36 in a separate step. To transfer documents from DW/3 to DW/36, make sure the documents are saved as RFT (revisable form text). The documents then can be uploaded with PC Support/36 and read by DW/36.

Documentation

CHAPTER

7

7

Cross-Referencing Files, Programs, and Procedures

by Ray Mueller

program by Paul Michels

Code on diskette:
Procedure XREF
RPG programs XREF01, XREF02, XREF03, XREF04, XREF05

This utility creates and prints four cross-reference reports: File Label by Program, Program by File Label, Program by Procedure, and Procedure by File.

You've been asked "merely" to add one more field to a screen and a report. Of course, you'll have to store the additional data in the relevant file – a file that has no room for an additional field – but without hesitation, you reply "no problem." Your reply really means, "I am the great omniscient programmer. I'll simply expand the record length of the file." Naturally, you realize only later that other programs access the file and your task of "merely" adding another field becomes "track down every program that accesses the file so they can be modified to recognize the new field." Still feel like the omniscient programmer?

You will if you use the S/36 XREF utility – a file, program, and procedure cross-reference utility that provides four reports. The XREF utility not only answers the question, "What programs access file X?" but also identifies the procedures that load those programs.

Information for these cross-reference reports is derived from the procedure members of a user-specified library. The utility copies the procedure members into a work file and then extracts procedure, program, and file name from each procedure's OCL code and writes this information into an output file. The utility produces one output file record for each file referenced in each program. If a procedure member references the file by a file name, the XREF utility extracts the file's disk label from the OCL code. The cross-reference reports are then obtained from sorted versions of the output file.

The utility consists of procedure XREFand five RPG programs – XREF01, XREF02, XREF03, XREF04, and XREF05. The procedure prompts for user input, creates the work file, performs the necessary sorting, and generally directs the action of the programs. Program XREF01 writes the output file; the subsequent programs read the sorted file and produce the desired reports. The procedure uses no prompt screens because user input is minimal: users need only specify the library to be analyzed, the report(s) to be produced, and whether the job is to be submitted to the job queue.

Procedure XREF (Figute 7-1) begins by testing for adequate disk space for the several files that will be created. If 2,000 blocks of contiguous disk space are not available, the procedure will be canceled. You may want to modify the procedure if this estimate of disk space is too big or too small for the libraries you will be analyzing; you could even modify the procedure so that it estimates the space required.

In any case, if sufficient disk space is available, the procedure requests the name of the library to be analyzed. The procedure then checks whether that library is not found. If the library is found, the procedure displays a list of five report options, allowing you to select any one of four reports by specifying options 1 through 4 or to select all of them by specifying option 5. If you enter any number other than 1, 2, 3, or 4, the procedure uses option 5, the default. The procedure stores the report option as parameter 2. The only other user input is the option to submit the job to the job queue.

After the user input phase is complete, the procedure loads the $MAINT utility, which copies all the procedure members in the specified library into work file OCLFILE. This work file is then processed by program XREF01 to create output file XREFA, which is needed by the remaining programs. After program XREF01 creates file XREFA, the procedure either branches to one of three tags (TWO, THREE, or FOUR) or proceeds sequentially, depending on the report option contained in parameter 2. If parameter 2 contains the value 1 or 5, the procedure sorts file XREFA in file name/program name sequence. Program XREF02 processes that sorted file (addrout file XREFB) and generates the files/programs cross-reference listing. When program XREF02 finishes, the procedure continues sequentially (if parameter 2 equals 5) or branches to the END tag.

If parameter 2 contains any number other than 1 or 5, the preceding steps are bypassed and processing commences at the appropriate tag (i.e., TAG TWO for option 2, TAG THREE for option 3, and so on). The processing logic at each of these tags is similar to that of TAG ONE (i.e., the procedure sorts file XREFA and calls a report-writing program). The difference between the TAGs is simply the order in which file XREFA is sorted and the report-writing program is called (e.g., at TAG TWO, file XREFA is sorted in program name/file name sequence). The procedure terminates by deleting file XREFA. Note that file XREFB is defined with RETAIN-J.

That's all there is to the procedure. The real heart of this utility, however, is program XREF01, so let's look at it in detail. Again, program XREF01 (Figure 7-2) reads file OCLFILE and creates file XREFA, which is sorted appropriately and then processed by the succeeding programs to produce the specific reports. With the input record stored as an array, the program can scan the record one byte at a time and extract program names, which are then stored in array PGN, and file names, which are stored in array LBL.

The program recognizes two types of input records (lines 7 and 12). Type 01 records contain two slashes followed by a blank in the first three positions and no asterisk in the fourth position. The // signifies the record may contain a procedure, file name, file label, or program name (a potentially informative record), but an asterisk would signify a prompt statement. A record without the // or with an asterisk in position 4 is designated as type 02. Because type 02 records contain no useful information for program XREF01, the first of the C-specs directs program logic to the end of the program if indicator 02 is on, and the RPG cycle resumes with the next record from file OCLFILE.

Provided the record is potentially informative (i.e., type 01), the program determines whether it is a // COPY statement. If the record is a // COPY statement, the program needs only to save the procedure name (which should be in positions 24 through 31) in field PRC. If the record is not a COPY statement, the record may be a LOAD or a FILE statement; thus, control passes to the ELSE statement (line 22), and the procedure calls subroutine LOAD to begin analyzing the record.

The ensuing processing sequence is illustrated in Figure 7-3. Subroutine LOAD tests for the presence of a LOAD statement and, if found, stores the program name associated with it. If the record is not a LOAD statement, subroutine LOAD calls subroutine FILE, which tests for the presence of a FILE statement. If the record is a FILE statement, subroutine FILE calls subroutine LABEL, which tests for the presence of a file label. If subroutine LABEL finds a file label, the file label is stored, as long as it is not a substitution expression. If there is no file label, subroutine LABEL calls subroutine NAME, which extracts a file name from the record, again, as long as it is not a substitution expression.

Note: Program XREF01 has no way of knowing what value will be substituted at execution time. That is why it is coded to drop any names or labels containing question marks. You can modify the program to test for any character, including the apostrophe, when looking for names or labels.

All of the subroutines operate much the same way as subroutine LOAD, so a description of the flow and function of the LOAD subroutine will help you understand the function of the other subroutines. Subroutine LOAD (lines 30 through 57 in Figure 7-2) sets on indicator 03 to signal that the keyword LOAD has been found; thus, the subroutine begins by setting off indicator 03. Next, field X is initialized. Field X serves as the index for array OCL as the subroutine scans each character of the input record. Field X also serves as the counter for the DO loop that performs the scanning (lines 33 through 54).

Within this DO loop, field X is incremented, and the current character of array OCL is tested against the character L (i.e., the subroutine is looking for the keyword LOAD). If the current character is not an L, control passes to the END statement in line 53, and the loop is repeated to examine the next element of array OCL (provided field X is less than or equal to 110). If the character is an L, the subroutine determines whether the six characters (beginning with the character before the L) are ƀLOADƀ. If the literal ƀLOADƀ is not found, control passes to the END statement in line 52, and, again, the loop is repeated. If the literal ƀLOADƀ is found, the subroutine sets on indicator 3 (line 40) and extracts the program name from the LOAD statement via another DO loop.

Before this DO loop begins, the program increments field X (the index for array OCL, which is currently positioned at the L) so that it points to the assumed start of the program name. Notice that the subroutine assumes that on the LOAD statement (i.e., array OCL), the word LOAD will be followed by a single blank and then the program name. With index X in the correct position, the program name is copied from array OCL into array PGN, one character at a time, in the subsequent DO loop (lines 44 through 50). If the DO loop encounters a blank or a comma, indicator 10 will terminate the DO loop because the DO loop's END statement is conditioned by indicator 10 (line 50).

Recall that subroutine LOAD sets on indicator 03 when the LOAD keyword is found. Thus, if the entire OCL statement has been scanned, and this keyword is not found, indicator 03 will not be on. In this situation, subroutine LOAD calls subroutine FILE (line 56). The call to subroutine FILE initiates the remainder of the subroutine control logic shown in Figure 7-2. As mentioned above, the remaining subroutines work in much the same way as subroutine LOAD does, scanning array OCL for the first letter of a keyword (i.e., F for FILE, L for LABEL, N for NAME) and then testing the array for the complete keyword. As you will see, the procedure and program names are not output at this point. Instead, they are saved until the file names that correspond to the procedure and program are found. When the program finds a file name, the program writes an output record that contains a procedure name, program name, and a file name or label name (lines 63 through 83 in Figure 7-2).

When you look closely at subroutine FILE, you will notice that this output is controlled by an EXCPT statement (line 78). Thus, subroutine FILE writes to the output file only if field LABEL is nonblank. (Field LABEL is filled by either subroutine NAME or subroutine LABEL, depending on whether a file name or a file label is found.) Thus, the subroutine produces one output record for each file referenced in the procedure member. Because there should be at least one label or file name for each procedure, each record in the output file contains all the necessary information for the reports to be generated.

Using the normal RPG cycle, program XREF01 continues processing until all the records in file OCLFILE have been processed. Again, through the exception output logic in subroutine FILE, the program writes one output record for each file referenced by each program. All that remains now is to sort file XREFA according to the cross-reference report desired and to write the reports. Although all the *hard* work is finished, let's look at a report-writing program to see how the remaining work is done. Because each of the report-writing programs are quite similar, a close look at program XREF02 will show you how the others function.

The purpose of program XREF02 (Figure 7-4) is to produce the files/programs cross-reference report (an example is shown in Figure 7-5). The program uses three files: file XREFA, which you are quite familiar with by now; file XREFB, which is the addrout file produced by the SORT utility called by procedure XREF; and file CROSSREF, a printer output file.

To get a better idea of the program's logic, keep in mind how the output is formatted (Figure 7-5) and how the program "sees" the input file (Figure 7-6). When you look at how the output file is sorted, think "control break," and you'll easily see how the program's output logic works. Specifically, a collection of program names are saved and printed when a file name control break occurs. Notice that in the report itself, up to seven program names are listed on each line — a more aesthetic output format than simply listing a column of file names and program names. This format also produces output that fits easily into a favorite storage medium — a three-ring binder. Now let's see how the program reads the input file and formats the output file.

Program XREF02 defines three arrays: array AXR, which consists of seven elements of six bytes each (these elements are names of programs that reference a given file; the array represents one line on the report), and arrays HED1 and HED2, which are defined at the end of the source listing.

The program's I-specs define three fields. The first is field CHR, the first character of the program name. The other two are the program name field, PGN, and the file name field, FLE, both of which define the control breaks mentioned above. The final I-specs define field LIB from the user LDA, which contains the name of the library being processed, as written by procedure XREF.

Program XREF02's C-specs copy the file name into field HFLE whenever a new file name is encountered (i.e., a level 2 control break occurs). Each time a new program name is encountered (i.e., a level 1 control break occurs), the program name is added to the end of array AXR, with field X acting as the array index. Notice, however, that if the first character of the program name is $ or #, either of which would indicate an IBM-supplied program, the program name is not output. The program assumes that you are not interested in file references by IBM-supplied programs, a reasonable assumption because you probably won't modify the program anyway.

When seven program names have been accumulated, the array of file names is output as exception time output (lines 24 and 25). If there are not seven file names to output, the contents of array AXP are output as exception time output when a new file name is encountered (lines 29 through 32). The comparison in line 29 prevents a line from printing with just a file name and no program names.

Notice that in the O-specs for this exception time output, the fields are blanked out after they are printed. By blanking the fields, the program prevents program names from spilling over from previous lines. For example, if there were 12 program names associated with a file name, the first seven would print on one line and the next five would print on the next line. If the output fields were not blanked out, the last two program names from the first line would appear as the last two program names on the second line. Because the file name also is blanked out, it will appear only once, even though there may be more than one line of program names for that given file name.

That's about all there is to program XREF02. Program XREF03 (Figure 7-7) is almost identical to program XREF02, except the order of file and program names are reversed because program XREF03's purpose is to print the programs/files cross-reference report (Figure 7-8). Programs XREF04 and XREF05 (Figures 7-9 and 7-10) should also be easy to follow because they mirror the structure of programs XREF02 and XREF03; programs XREF04 and XREF05 create files/procedure cross-reference reports (Figure 7-11) and programs/procedures cross-reference reports (Figure 7-12), respectively. The only significant departure from the logic in program XREF02 is that program XREF04 does not test for IBM-supplied procedures (i.e., procedure names that begin with $ or #); the program assumes that you want to know which files are used in each procedure, including files being processed by an IBM utility such as #GSORT.

The XREF utility can be a real productivity boost, but it is not designed to handle every variation of OCL programming. Most significantly, this utility cannot handle substitution expressions; you'll either have to manually resolve the substitution expressions before you run the utility or modify the utility to simulate the runtime substitution. The utility also does not work properly if FILE statements are not coded between the LOAD and RUN statements.

Extra spaces between the keyword FILE and the file name or between the keyword LOAD and the program name cause problems, too. These spaces may make your programs more readable, but the XREF utility is not designed to recognize them . If your procedures contain additional spaces, you could modify the program to scan for the first nonblank character following the keyword instead of simply skipping a given number of spaces. If you use the plus sign (+) as a continuation character in your OCL statements, you'll also have to modify the utility to skip over any plus signs between a keyword and the program or file name.

The last limitation in using the XREF utility concerns procedure calls from different libraries. If a procedure calls programs from another library, you will not get the file name cross-references for that program. You can bypass this limitation in one of two ways. First, you could copy all of your libraries that contains OCL into file OCLFILE. If this solution strains your system, you could write a program that has the necessary logic to process multiple libraries.

Despite these few special-case limitations, the XREF utility can be indispensable when you need to track down the connections between files, programs, and procedures. When you have the XREF utility on your system, you'll still look like "the great omniscient programmer" when someone requests a "simple" expansion of file records.

Figure 7-1

Procedure XREF

```
** LIBRARY CROSS-REFERENCE PROCEDURE
// REGION SIZE-04
// IF JOBQ-YES   GOTO START
*
*  Test for disk work space
*
// IF BLOCKS-2000 GOTO ENOUGH
// * ' There is not enough contiguous disk space to run this'
// * ' procedure   Need at least 2000 blocks '
```

```
// PAUSE
// CANCEL
// TAG ENOUGH
*
// * ' Library cross-reference procedure is running'
// IF ?1?-  * ' Enter library name to process'
// IFF DATAF1-?1R? PAUSE ' Library ?1? is not on disk - procedure will end'
// IFF DATAF1-?1?  CANCEL
// LOCAL OFFSET-1,DATA-'?1?',BLANK-8
*
*
*  Prompt for options
*
// * ' '
// * ' Library·  ?1?'
// * ' '
// * '   1)  Files and programs in which they are used'
// * '   2)  Programs and file labels accessed by them'
// * '   3)  Files and procedures in which they are used'
// * '   4)  Programs and procedures that load them'
// * '   5)  ALL reports'
// * ' '
// * ' Enter report option desired.  Default is ALL reports'
*
// IF ?2R?-  EVALUATE P2-5
// IF ?2?-0  EVALUATE P2-5
// IF ?2?>5  EVALUATE P2-5
*
// * ' '
// * ' Option ?2? selected.  Put on JOBQ?  (Y/N)'
// IFF ?3R?-Y    GOTO START
//               JOBQ 3,?CLIB?,XREF,?1?,?2?
//               RETURN
*
//               TAG START
*
*  Create OCL workfile
*
// IF JOBQ-NO    * ' Getting procedure members from ?1? library'
// LOAD $MAINT
// FILE NAME-OCLFILE,RETAIN-J,BLOCKS-2000
// RUN // COPY FROM-?1?,TO-DISK,FILE-OCLFILE,LIBRARY-P,RECL-120,NAME-ALL
// END
// IF JOBQ-NO    * ' Creating OCL statement workfile'
// IF DATAF1-XREFA DELETE XREFA,F1
// LOAD XREF01
// FILE NAME-OCLFILE,RETAIN-S
// FILE NAME-XREF,LABEL-XREFA,RECORDS-3000
// RUN
*
*  Determine which reports to print
*
// IF ?2?-2 GOTO TWO
// IF ?2?-3 GOTO THREE
// IF ?2?-4 GOTO FOUR
*
// IF JOBQ-NO * ' Files / programs cross-reference is running'
// REGION SIZE-36
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-XREFA
// FILE NAME-OUTPUT,LABEL-XREFB,RETAIN-J,RECORDS-?F'A,XREFA'?
// RUN
   HSORTA    14A        3    3
   FNC   7  14                        FILE NAME
   FNC   1   6                        PROGRAM NAME
// END
// LOAD XREF02
// FILE NAME-XREFA
// FILE NAME-XREFB,RETAIN-S
// RUN
// IFF ?2?-5  GOTO END
*
// TAG TWO
// IF JOBQ-NO * ' Programs / files cross-reference is running'
// REGION SIZE-36
```

```
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-XREFA
// FILE NAME-OUTPUT,LABEL-XREFB,RETAIN-J,RECORDS-?F'A,XREFA'?
// RUN
     HSORTA    14A        3    3
     FNC   1   6                         PROGRAM NAME
     FNC   7  14                         FILE NAME
// END
// LOAD XREF03
// FILE NAME-XREFA
// FILE NAME-XREFB,RETAIN-S
// RUN
// IFF ?2?=5  GOTO END
*
// TAG THREE
// IF JOBQ-NO * ' Procedures / files cross-reference is running'
// REGION SIZE-36
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-XREFA
// FILE NAME-OUTPUT,LABEL-XREFB,RETAIN-J,RECORDS-?F'A,XREFA'?
// RUN
     HSORTA    19A        3    3
     FNC   7  14                         FILE NAME
     FNC  15  22                         PROCEDURE NAME
// END
// LOAD XREF04
// FILE NAME-XREFA
// FILE NAME-XREFB,RETAIN-S
// RUN
// IFF ?2?=5  GOTO END
*
// TAG FOUR
// IF JOBQ-NO * ' Programs / procedures cross-reference is running'
// REGION SIZE-36
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-XREFA
// FILE NAME-OUTPUT,LABEL-XREFB,RETAIN-J,RECORDS-?F'A,XREFA'?
// RUN
     HSORTA    14A        3    3
     FNC   1   6                         PROGRAM NAME
     FNC  15  22                         PROCEDURE NAME
// END
// LOAD XREF05
// FILE NAME-XREFA
// FILE NAME-XREFB,RETAIN-S
// RUN
*
// TAG END
DELETE XREFA,F1
```

Figure 7-2

Program XREF01

```
*         1         2         3         4         5         6         7         8
0001 H    P 64                        B                                    XREF01
0002 FOCLFILE IPE F1200 120    2        DISK
0003 FXREF    O   F 880  22    2        DISK
0003AE                     CHKCPY  1    1 23                //  COPY LIBRARY-P,NAME-
0004 E                     LBL          8  1                FILE NAME OR LABEL
0005 E                     PGN          6  1                PROGRAM NAME
0006 E                     OCL        120  1                OCL STATEMENT
0007 IOCLFILE NS  01   1 C/   2 C/   3 C
0008 I       AND       4NC*
0009 I                                       1  23 CHK
0010 I                                       1 120 OCL
0011 I                                      24  31 NAM
0012 I        NS  02
0013 I            DS
0014 I                                       1   8 LABEL
0015 I                                       1   8 LBL
0016 C*
0017 C*
0018 C   02                 GOTO THRU
0019 C          CHK         IFEQ CHKCPY   '                     TEST FOR NEW PROCEDURE
0020 C                      MOVE NAM       PRC     8            SAVE PROCEDURE NAME
0021 C                      GOTO THRU                           SKIP THE REST OF THIS CYCLE
```

```
C                     ELSE                                  IF NOT "// COPY " STATEMENT
C                     EXSR LOAD                             LOOK FOR LOAD STATEMENT
C                     END
C           THRU      TAG
C*
C*  Test to see if current statement is a LOAD statement
C*    . if it is, extract the program name
C*
C           LOAD      BEGSR
C                     SETOF                     03          WILL TURN ON IF "LOAD" STATMT FOUND
C                     Z-ADD0         X       30
C           X         DOUGT110                              SCAN THRU POSTION 110 OF STATEMENT
C                     ADD  1         X
C           OCL,X     IFEQ 'L'                              1ST LETTER OF "LOAD"   MAKE TEST
C                     SUB  1         X                      DECREMENT 1 TO INCL BLANK IN TEST
C                     MOVEAOCL,X     TEST6   6              EXTRACT 6 TEST CHARACTERS
C                     ADD  1         XRESTORE INDEX
C           TEST6     IFEQ ' LOAD '                         TEST FOR "LOAD" KEYWORD
C                     SETON                     03          03 ON - CURRENT RECORD IS LOAD STMT
C                     ADD  5         X                      INCRMN INDEX BY 5 TO GET PGM NAME
C                     Z-ADD1         Y       10             INITIALIZE PROGRAM NAME ARRAY INDEX
C                     MOVE *BLANK    PGN                    CLEAR PROGRAM NAME ARRAY
C           Y         DOUGT6                                GET UP TO 6 CHARACTERS FOR PGM NAME
C                     MOVE OCL,X     PGN,Y                  MOVE CHARACTER TO PGM NAME ARRAY
C                     ADD  1         X
C                     ADD  1         Y
C           OCL,X     COMP ','                      10  COMMA OR BLANK INDICATE END OF THE
C  N10      OCL,X     COMP ' '                      10  PROGRAM NAME
C   10                GOTO EXIT1                            NAME FOUND - EXIT SUBROUTINE
C                     END
C                     GOTO EXIT1
C                     END
C                     END
C                     END
C           EXIT1     TAG                                   IF CURRENT RECORD WAS NOT A LOAD
C  NO3                EXSR FILE                             STATEMENT, TEST FOR A FILE STATEMNT
C                     ENDSR
C*
C*  Test to see if current record is a FILE statement
C*     same basic logic is used in following subroutines
C*  as in LOAD subroutine
C*
C           FILE      BEGSR
C                     MOVE *BLANK    LBL
C                     Z-ADD0         X
C           X         DOUEQ108
C                     ADD  1         X
C           OCL,X     IFEQ 'F'
C                     SUB  1         X
C                     MOVEAOCL,X     TEST6
C                     ADD  1         X
C           TEST6     IFEQ ' FILE '                         IS CURRENT RECORD IS FILE STATEMENT
C                     EXSR LABEL                            LOOK FOR "LABEL" KEYWORD
C           LABEL     IFEQ *BLANKS                          IF "LABEL" IS NOT FOUND,
C                     EXSR NAME                                GET THE "NAME"
C                     END
C           LABEL     IFNE *BLANKS                          IF LABEL OR NAME FOUND WRITE RECORD
C                     EXCPTWRITE
C                     END
C                     END
C                     END
C                     END
C                     ENDSR
C*
C*  Test for keyword LABEL in the OCL FILE statement
C*
C           LABEL     BEGSR
C                     Z-ADD0         X
C           X         DOUEQ107
C                     ADD  1         X
C           OCL,X     IFEQ 'L'
C                     SUB  1         X
C                     MOVEAOCL,X     TEST7   7
C                     ADD  1         X
C           TEST7     IFEQ ',LABEL-'
```

```
0097 C                    ADD  6         X
0098 C                    Z-ADD1         Y
0099 C           Y        DOUGT8
0100 C           OCL,X    IFEQ '?'                            EXCLUDE LABELS CONTAINING
0101 C                    MOVE *BLANKS   LABEL                SUBSTITUTION EXPRESSIONS
0102 C                    GOTO EXIT2
0103 C                    END
0104 C                    MOVE OCL,X     LBL,Y
0105 C                    ADD  1         X
0106 C                    ADD  1         Y
0107 C           OCL,X    COMP ','                        04
0108 C  N04      OCL,X    COMP ' '                        04
0109 C   04               GOTO EXIT2
0110 C                    END
0111 C                    GOTO EXIT2
0112 C                    END
0113 C                    END
0114 C                    END
0115 C           EXIT2    ENDSR
0116 C*
0117 C*  If LABEL keyword not found, use file NAME for output
0118 C*
0119 C           NAME     BEGSR
0120 C                    Z-ADD0         X
0121 C           X        DOUEQ108
0122 C                    ADD  1         X
0123 C           OCL,X    IFEQ 'N'
0124 C                    SUB  1         X
0125 C                    MOVEAOCL,X     TEST6
0126 C                    ADD  1         X
0127 C           TEST6    IFEQ ' NAME-'
0128 C                    ADD  5         X
0129 C                    Z-ADD1         Y        10
0130 C           Y        DOUGT8
0131 C           OCL,X    IFEQ '?'                            EXCLUDE FILE NAMES CONTAINING
0132 C                    MOVE *BLANKS   LABEL                SUBSTITUTION EXPRESSIONS
0133 C                    GOTO EXIT3
0134 C                    END
0135 C                    MOVE OCL,X     LBL,Y
0136 C                    ADD  1         X
0137 C                    ADD  1         Y
0138 C           OCL,X    COMP ','                        04
0139 C  N04      OCL,X    COMP ' '                        04
0140 C   04               GOTO EXIT3
0141 C                    END
0142 C                    GOTO EXIT3
0143 C                    END
0144 C                    END
0145 C                    END
0146 C           EXIT3    ENDSR
0147 C*
0148 C*
0149 OXREF    E                WRITE
0150 O                         PGN        6
0151 O                         LBL       14
0152 O                         PRC       22
**
// COPY LIBRARY-P,NAME-
```

Figure 7-3

Flow chart for calling subroutines in program XREF01

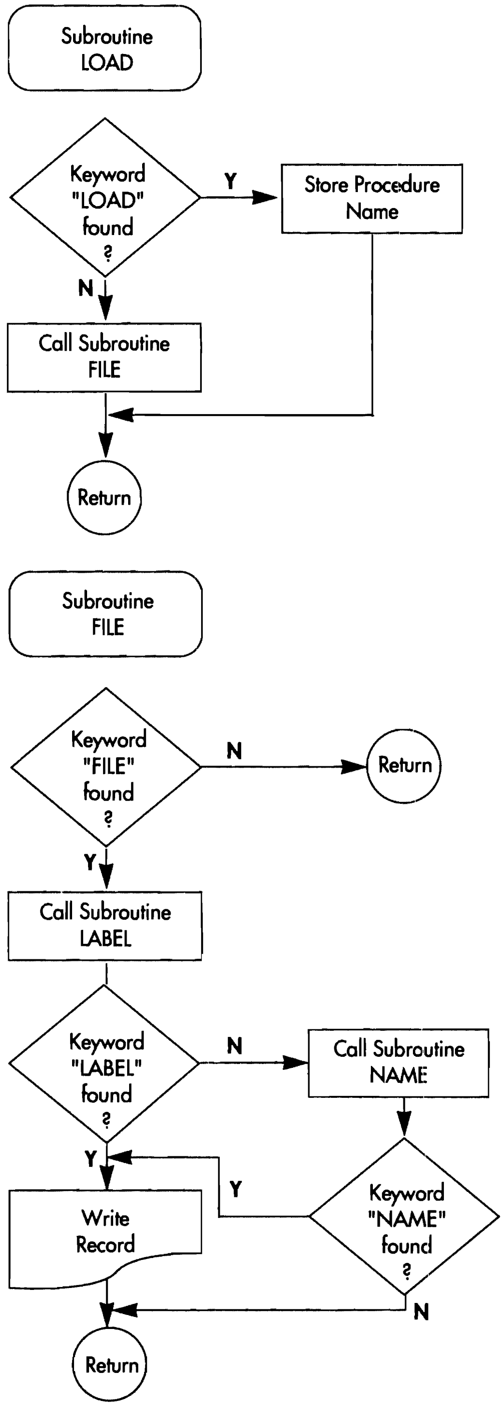


Figure 7-4

Program XREF02

```
*. .   . 1  .      2         3  .. ..  4         5 .   . 6 .       7    . . . 8
0001 H    P 64                        B        1                          XREF02
0002 F* PRINT FILES/PROGRAMS XREF
0003 FXREFA    IP  F  22  22R  I         DISK
0004 FXREFB    IRE F 300   3  3IT       EDISK
0005 FCROSSREFO    F  80  80     OF      PRINTER
0006 E                      AXR        7  6
0007 E                      HED1   32  32  1
0008 E                      HED2   48  48  1
0009 E    XREFB   XREFA
0010 IXREFA   NS
0011 I                                        1   1 CHR
0012 I                                        1   6 PRG    L1
0013 I                                        7  14 FLE    L2
0014 I            UDS
0015 I                                        1   8 LIB
0016 C*
0017 C*
0018 C    L2                 MOVE FLE       HFLE     B
0019 C    L1       CHR       COMP '$'                      10  TEST FOR IBM UTILITY
0020 C    L1N10    CHR       COMP '#'                      10  TEST FOR IBM SORT PROGRAM
0021 C    L1 10              GOTO BYPASS                       BYPASS, IF IBM PROGRAM
0022 C    L1                 ADD  1         X        10
0023 C    L1                 MOVE PRG       AXR,X
0024 C    L1       X         COMP 6                   99
0025 C    L1 99              EXCPT 0026 C   L1 99             SUB  X          X
0027 C    L1                 SETOF                    99
0028 C             BYPASS    TAG
0029 CL2           X         COMP 0                   99
0030 CL2 99                  EXCPT
0031 CL2 99                  SUB  X         X
0032 CL2                     SETOF                    99
0033 C*
0034 C*
0035 OCROSSREFH  102    OF
0036 O        OR        1P
0037 O                         HED1      44
0038 O                         UDATE Y   53
0039 O                                   67 'PAGE'
0040 O                         PAGE      72
0041 O         H  2     OF
0042 O        OR        1P
0043 O                                   20 'LIBRARY''
0044 O                         LIB       29
0045 O         H  2     OF
0046 O        OR        1P
0047 O                                   22 'FILE LABEL'
0048 O                         HED2      72
0049 O         E  1     99
0050 O                         HFLE    B 20
0051 O                         AXR,1   B 30
0052 O                         AXR,2   B 37
0053 O                         AXR,3   B 44
0054 O                         AXR,4   B 51
0055 O                         AXR,5   B 58
0056 O                         AXR,6   B 65
0057 O                         AXR,7   B 72
**
FILES / PROGRAMS CROSS REFERENCE
**
——— PROGRAMS THAT USE FILE ———
```

Figure 7-5

Example file/program cross-reference report

```
------------ PROGRAMS THAT USE FILE ------------
AR11     ARDSL    ARJRN    AROPN    ARRAA    CTCAD    CTPRM
         CTSCO    CTSMN    INWRK
AR11A    ARRAA    CTCAD    CTSMN
AR11D    AROPM    CTAAC    CTCAD    CTPRM    CTSMN
AR11L    ARBDO    CTCAD    CTPRM
AR11R    ARBDJ    ARBDO    ARBDR    ARDSL    CTCAD    CTPRM
         CTSCO    CTSMN
```

Figure 7-6

Sorted version of output file

```
program            file
  AR11             ADSL
  AR11             ARJN
  AR11             AROPN
   .                .
   .                .
   .                .
  AR11             INWRK
  AR11A            ARRAA
  AR11A            CTCAD
  AR11A            CTSMN
  AR11D            AROPM
  AR11D            CTAAC
  AR11D            CTCAD
  AR11D            CTPRM
  AR11D            CTSMN
  AR11L            ARBDO
   .                .
   .                .
   .                .
```

Figure 7-7

Program XREF03

```
*        1         2         3         4         5         6         7         8
0001 H    P 64                      B       1                             XREF03
0002 F*  PRINT PROGRAM/FILES XREF
0003 FXREFA   IP  F  22  22R  I        DISK
0004 FXREFB   IRE F 300  3  3IT      EDISK
0005 FCROSSREFO   F  80  80     OF     PRINTER
0006 E                    AXR         6  8
0007 E                    HED1   32  32  1
0008 E                    HED2   51  51  1
0009 E    XREFB   XREFA
0010 IXREFA   NS
0011 I                                        1   1 CHR
0012 I                                        1   6 PRG   L2
0013 I                                        7  14 FLE   L1
0014 I           UDS
0015 I                                        1   8 LIB
0016 C*
0017 C*
0018 C           CHR       COMP '$'                      10
0019 C  N10      CHR       COMP '#'                      10
0020 C    10               GOTO BYPASS
0021 C    L2               MOVE PRG       HPRG    6
0022 C    L1     X         ADD  1         X       20
0023 C    L1               MOVE FLE       AXR,X
0024 C    L1     X         COMP 5                    99
0025 C    L1 99            EXCPT
0026 C    L1 99  X         SUB  X         X
0027 C    L1               SETOF                     99
0028 C           BYPASS    TAG
0029 CL2         X         COMP 0                    99
0030 CL2 99                EXCPT
0031 CL2 99      X         SUB  X         X
0032 CL2                   SETOF                     99
0033 C*
0034 C*
0035 OCROSSREFH  102   OF
0036 O       OR        1P
0037 O                         HED1      44
0038 O                         UDATE Y   53
0039 O                                   67 'PAGE'
0040 O                         PAGE      72
0041 O        H  2     OF
0042 O       OR        1P
0043 O                                   20 'LIBRARY· '
0044 O                         LIB       29
0045 O        H  2     OF
0046 O       OR        1P
```

```
0047 O                               19 'PROGRAM'
0048 O                        HED2      75
0049 O         E  1      99
0050 O                        HPRG   B  18
0051 O                        AXR,1  B  32
0052 O                        AXR,2  B  41
0053 O                        AXR,3  B  50
0054 O                        AXR,4  B  59
0055 O                        AXR,5  B  68
0056 O                        AXR,6  B  77
**
PROGRAMS / FILES CROSS REFERENCE
**
——— FILE LABELS USED IN PROGRAM ———
```

Figure 7-8

Example program/file cross-reference report

```
----------- FILE LABELS USED IN PROGRAM -----------

CTAVP     CT14G     CT14H
CTBCK     IN01K     IN01L     IN01M
CTBIL     CT30C     CT30G
CTCAD     AR11      AR11A     AR11D     AR11L     AR11R     AR11X     AR60C
          AR60P     AR61C     AR61E     AR61ES    AR61G     AR61H     AR61J
          AR95      CT01      CT02      CT10A     CT10D     CT10E     CT10G
          CT10P     CT10Q     CT10T     CT11      CT13      CT14J     CT14M
          CT17B     CT18/     CT18C     CT18D     CT18F     CT20A     CT20H
          CT20J     CT20N     CT20R     CT20V     CT20W     CT20Z     CT25D
```

Figure 7-9

Program XREF04

```
*.     . 1 .     2    .  3        4        5    .   6        7  .  . 8
0001 H    P 64                         B       1                        XREF04
0002 F*  PRINT FILES/PROCEDURES XREF
0003 FXREFA    IP  F  22  22R  I          DISK
0004 FXREFB    IRE F 300   3  3IT        EDISK
0005 FCROSSREFO    F  80  80      OF      PRINTER
0006 E                        AXR           6  8
0007 E                        HED1    34   34  1
0008 E                        HED2    51   51  1
0009 E     XREFB    XREFA
0010 IXREFA    NS
0011 I                                          15  22 PRC    L1
0012 I                                           7  14 FLE    L2
0013 I              UDS
0014 I                                           1   8 LIB
0015 C*
0016 C*
0017 C    L2                   MOVE FLE       HFLE     8
0018 C    L1        X          ADD  1         X       20
0019 C    L1                   MOVE PRC       AXR,X
0020 C    L1        X          COMP 5                      99
0021 C    L1  99               EXCPT
0022 C    L1  99    X          SUB  X         X
0023 C    L1                   SETOF                       99
0024 CL2            X          COMP 0                      99
0025 CL2  99                   EXCPT
0026 CL2  99        X          SUB  X         X
0027 CL2                       SETOF                       99
0028 C*
0029 C*
0030 OCROSSREFH  102    OF
0031 O         OR        1P
0032 O                        HED1      46
0033 O                        UDATE Y   56
0034 O                                  67 'PAGE'
0035 O                        PAGE      72
0036 O         H  2     OF
0037 O         OR        1P
0038 O                                  20 'LIBRARY'
0039 O                        LIB       29
0040 O         H  2     OF
0041 O         OR        1P
0042 O                                  16 'FILE'
0043 O                        HED2      75
0044 O         E  1      99
0045 O                        HFLE   B  20
```

```
O                          AXR,1  B  32
O                          AXR,2  B  41
O                          AXR,3  B  50
O                          AXR,4  B  59
O                          AXR,5  B  68
O                          AXR,6  B  77
**
FILES / PROCEDURES CROSS REFERENCE
**
------ PROCEDURES THAT USE FILE ------
```

Figure 7-10

Program XREF05

```
*        1         2         3         4    .    5         6         7         8
H    P 64                          B       1                              XREF05
F*  PRINT PROGRAMS/PROCEDURES XREF
FXREFA   IP  F  22   22R  I        DISK
FXREFB   IRE F 300    3  3IT      EDISK
FCROSSREFO   F  80   80     OF      PRINTER
E                     AXR         6  8
E                     HED1   37  37  1
E                     HED2   51  51  1
E    XREFB    XREFA
IXREFA    NS
I                                         1   1 CHR
I                                         1   6 PROG  L2
I                                        15  22 PROC  L1
I           UDS
I                                         1   8 LIB
C*
C*
C           CHR       COMP '$'                         10
C  N10      CHR       COMP '#'                         10
C   10                GOTO BYPASS
C   L2                MOVE PROG      HPROG   6
C   L1                ADD  1         X       20
C   L1                MOVE PROC      AXR,X
C   L1      X         COMP 5                    99
C   L1 99             EXCPT
C   L1 99             SUB  X         X
C   L1                SETOF                     99
C           BYPASS    TAG
CL2         X         COMP 0                    99
CL2 99                EXCPT
CL2 99                SUB  X         X
CL2                   SETOF                     99
OCROSSREFH  102   OF
O       OR        1P
O                          HED1      49
O                          UDATE Y   56
O                                    67 'PAGE'
O                          PAGE      72
O        H  2      OF
O       OR        1P
O                                    20 'LIBRARY:'
O                          LIB       29
O        H  2      OF
O       OR        1P
O                                    19 'PROGRAM'
O                          HED2      75
O        E  1      99
O                          HPROG  B  20
O                          AXR,1  B  32
O                          AXR,2  B  41
O                          AXR,3  B  50
O                          AXR,4  B  59
O                          AXR,5  B  68
O                          AXR,6  B  77
**
PROGRAMS / PROCEDURES CROSS REFERENCE
**
------ PROCEDURES THAT LOAD PROGRAM ------
```

Figure 7-11

Example program/procedure cross-reference report

```
---------- PROCEDURES THAT LOAD PROGRAM -----------

CTAD1     CTP20Z    CTP41     CTXN2     CTXX25
CTAD7     CTP17
CTAIR     CTP22     CTP25J    CTP30
CTAIRT    CTP22     CTP25J    CTP30
CTALT     CTX91
CTAVL     CTP11     CTP13     CTP14E    CTP14F    CTP14L    CTP14M
          CTX10A    CTX14
```

Figure 7-12

Example procedure/file cross-reference report

```
------------ PROCEDURES THAT USE FILE -------------

AR11      ARP11     INP04
AR11A     ARP11A
AR11D     ARP11D
AR11L     ARP11L
AR11R     ARP11R
```

Cross-Referencing Queries

by Gary T. Kratzer and Tim Gardner

program by Tim Gardner

Code on diskette:
Procedure QRYXRF
RPG program QRYXRF
Screen format member QRYXRFFM
RPG code #QRYEXT
Assembler subroutine SUBLR

System-wide documentation has always been the bane of the data processing manager, and with Query/36 and its wide range of uses, this task is even more difficult. Query/36 itself provides no "stock" facility for retrieving or listing query cross-referencing information. Utility QRYXRF, which assists Query users in developing cross-referencing information and listings of the queries on their systems, is especially useful for the avid Query user because it places the generated information in a file for which IDDU specifications are given. Thus, in addition to the sample report presented here, you can generate numerous other reports to assist you in documenting and listing your queries and the files and formats they use.

QRYXRF utility comprises a prompt screen (Figure 7-13), screen format member QRYXRFFM (Figure 7-14), RPG program QRYXRF (Figure 7-15), and procedure QRYXRF (Figure 7-16). Before using QRYXRF, you need to build the IDDU specifications (Figures 7-17 and 7-18) for the cross-reference file #QRYEXT generated by program QRYXRF. You may also want to add your own column headings and specify numeric editing for field LSTCHG (Date Last Changed). Note that file #QRYEXT contains a record for each file format the query uses, not just one record for each query. Rather than using IDDU to build the IDDU specifications, you can simply key the F- and I-specs for the file (Figure 7-19) into a source member and then use the S/36 IDDUXLAT procedure to translate them into their IDDU field, format, and file definitions.

Figure 7-13
QRYXRF prompt screen

```
                    QUERY CROSS-REFERENCE FILE BUILD
                                                        Optional *
 Library 1                                                       *

 Library 2                                                       *

 Library 3                                                       *

 Library 4                                                       *

 Library 5                                                       *

 Library 6                                                       *

 Library 7                                                       *

 Library 8                                                       *

Cmd3=Previous Menu        Cmd4=Put on Job Queue     Cmd7=End of Job
```

Utility QRYXRF in Action

When you run procedure QRYXRF, the prompt screen asks you to enter one to eight libraries that contain your queries. After entering the library names, you may either press Enter to continue processing interactively, or press Command key 4 to place the procedure on the job queue. In either case, procedure QRYXRF loads program QRYXRF, which uses subroutine SUBRLD to search the requested libraries for queries. (Queries are stored in libraries as subroutine members with a subtype of 58.) When the program finds a query, it uses subroutine SUBRLR to read the first two sectors of the member, where the cross-reference information about each query is stored. The program then formats this information and outputs it to file #QRYEXT.

When program QRYXRF has processed all the libraries, the program ends, and procedure QRYXRF runs the IDDULINK procedure that links the query definition you created earlier to the output file #QRYEXT. In the IDDULINK procedure call within procedure QRYXRF, be sure to specify the name of the folder where the query definition exists.

Putting Utility QRYXRF to Work

Now that you've created a file with information about the queries on your system, you can use Query/36 to generate a variety of cross-reference reports. The sample report in Figure 7-20 is part of a format cross-reference listing (Figures 7-21 and 7-22 show the field selection and sort sequence specification for the generated report), which can be useful when the IDDU specifications for a file format change. When you change a file format, it is often necessary to update the query so that system error message QRY-1058, "File level does not match query," does not occur the next time the query is run (usually during a batch job in the middle of the night).

With some minor program modifications, you can extend utility QRYXRF to search more libraries or to create a different output file for each library. You can generate countless other Query reports from file #QRYEXT. It would be helpful, for example, to examine all queries that reference a customer master file. So put utility QRYXRF to work for you to lighten the burden of system-wide documentation.

Figure 7-14

Screen format member QRYXRFFM

```
*      1           2         3         4          5         6         7       8
     SPARAMTRS           YY                                         DCG
     D         32 125Y                                      CQUERY CROSS-REFERENCE FX
     DILE BUILD
     D         10 269Y                                      COptional *
     D         63 6 2Y                                      CLibrary 1            X
     D
     DPARM01    8 66701 Y     41 Y         41  Y
     D          1 678Y                                      C*
     D         63 8 2Y                                      CLibrary 2            X
     D
     DPARM02    8 86702 Y     42 Y         42  Y
     D          1 878Y                                      C*
     D         6310 2Y                                      CLibrary 3            X
     D
     DPARM03    8106703 Y     43 Y         43  Y
     D          11078Y                                      C*
     D         6312 2Y                                      CLibrary 4            X
     D
     DPARM04    8126704 Y     44 Y         44  Y
     D          11278Y                                      C*
     D         6314 2Y                                      CLibrary 5            X
     D
     DPARM05    8146705 Y     45 Y         45  Y
     D          11478Y                                      C*
     D         6316 2Y                                      CLibrary 6            X
     D
     DPARM06    8166706 Y     46 Y         46  Y
     D          11678Y                                      C*
     D         6318 2Y                                      CLibrary 7            X
     D
     DPARM07    8186707 Y     47 Y         47  Y
     D          11878Y                                      C*
     D         6320 2Y                                      CLibrary 8            X
     D
     DPARM08    8206708 Y     48 Y         48  Y
     D          12078Y                                      C*
     D          422 2Y                 Y                    CCmd3
     D         1322 7Y                                      CPrevious Menu
     D          42228Y                 Y                    CCmd4
     D         162234Y                                      CPut on Job Queue
     D          42254Y                 Y                    CCmd7
     D         102259Y                                      CEnd of Job
     DMESSG    60242009                09
```

Figure 7-15

Program QRYXRF

```
*    1   . 2  .   3 . .   4  .   5 . .   6    . 7    . . 8
H      64                  B      !                 QRYXRF
F*****************************************************************
F* PROGRAM NAME.      QRYXRF
F* DESCRIPTION ..     Build an extract file of query formats
F* PROGRAMMER. .  . . Tim Gardner
F* DATE WRITTEN. .  . May 1990
F*****************************************************************
FQRYEXT  O   F 108              DISK
E*****************************************************************
E                    FIL       5 54               Query File OS's
E                    LIB       8  8               Input libraries
I*****************************************************************
I           UDS
I                                        1  64 LIB
I*
IDIRDS      DS
I                                        1   1 DRTYPE
I                                        2   9 DRNAME
I                                       10  15 DRADDR
I                                       16  18 DR#TXT
I                                       19  22 DRLINK
I                                       23  27 DR#STM
I                                       28  31 DRSCA
I                                       32  33 DRRLD
I                                       34  36 DRCORE
I                                       37  37 DRATR1
I                                       38  38 DRATR2
I                                       39  39 DRATR3
I                                       40  41 DRMRT
I                                       42  43 DRREL
I                                       44  46 DRTOTL
I                                       47  47 DRATR4
I                                       48  53 DRMOD
I                                       54  59 DRDATE
I                                       60  63 DRTIME
I                                       64  650DRATR5
I                                       66  69 DRPTF@
I                                       70  70 DRATR6
I                                       71  80 AVAIL
I*
I           DS
I                                        1 256 BUFF1
I                                      257 512 BUFF2
I                                        1   1 HEX00
I                                        2  45 QRYDSC
I                                       46  53 LSTUSR
I                                       73  73 DISKBT
I                                       75  75 PRTRBT
I                                      131 400 FIL
I*
I           DS
I                                        1  54 FILEDS
I                                        1   1 HEXZZ
I                                        1   8 FILNAM
I                                        9  16 DICNAM
I                                       17  24 IFILNM
I                                       36  43 IFMTNM
C*****************************************************************
C*
C                     BITON'01234567'HEXFF   1        Initialize Hex FF'S
C*
C           1         DO   8         L       10       Do max 8 libraries
C           LIB,L     IFNE *BLANKS                     If valid library
C*
C                     MOVE LIB,L     LIBNAM             Initialize for
C                     MOVE 'R'       MBRTYP             directory reads
C                     MOVE *BLANKS   MBRNAM             through all R-
C                     EXSR GETDIR                       modules
C*
C           MBRTYP    DOWEQ'R'                          Do while R-module
C*
C           DRATR5    IFEQ 58                            If Query
C                     EXSR OPEN                           Open it
C  NU1                EXSR READ                           Read it
C  NU1                EXSR QRYPRC                         Process it
```

```
C                    END                                    End IF
C*
C  NU1               EXSR GETDIR                            Get next dir
C                    END                                   End DOW
C                    END                                  End IF
C  NU1               END                                 End DO
C*
C                    SETON                        LR
C*****************************************************************
C* Open R-module for sector reads
C*
C          OPEN      BEGSR
C*
C                    MOVE '0'       OP                   Specify open
C                    EXIT SUBRLR                         Sector get
C                    RLABL          OP      1             Opcode
C                    RLABL          LIBNAM                Library
C                    RLABL          DRNAME                Member
C                    RLABL          MBRTYP                Type
C                    RLABL          RCODE   1             Return code
C*
C          RCODE     COMP '0'                     U1U1   Terminal error
C*
C                    ENDSR
C*****************************************************************
C* Read first 512 bytes of query definition
C*
C          READ      BEGSR
C*
C                    MOVE 'N'       OP                   Specify read next
C                    EXIT SUBRLR                         Sector get
C                    RLABL          OP                    Opcode
C                    RLABL          BUFF1                 Text buffer 1
C                    RLABL          RCODE                 Return code
C*
C          RCODE     IFEQ '0'                            If good return
C                    EXIT SUBRLR                          Sector get
C                    RLABL          OP                     Opcode
C                    RLABL          BUFF2                  Text buffer 2
C                    RLABL          RCODE                  Return code
C*
C                    ELSE                                Else
C                    SETON                        U1      Terminal error
C                    END                                 End IF
C*
C                    ENDSR
C*****************************************************************
C* Exit to SUBRLD for next member name or to reset library
C*
C          GETDIR    BEGSR
C*
C                    EXIT SUBRLD                         Directory read
C                    RLABL          LIBNAM  8             Library
C                    RLABL          MBRNAM  8             Member
C                    RLABL          MBRTYP  1             Type
C                    RLABL          DIRDS                 Directory DS
C                    RLABL          RCODE                 Return code
C*
C          RCODE     IFEQ '2'                            If end of R-modules
C                    MOVE 'Q'       MBRTYP                Flag to quit libr
C                    ELSE                                Else
C          RCODE     COMP '0'                     U1U1    Terminal error
C   U1               MOVE 'Q'       MBRTYP                Flag to quit libr
C                    END                                 End IF
C*
C                    ENDSR
C*****************************************************************
C* Evaluate query for output
C*
C          QRYPRC    BEGSR
C*
C          DISKBT    IFNE HEXFF                          Determine type
C                    MOVE 'F'       QTYPE   1             F=File
C                    ELSE
C          PRTRBT    IFNE HEXFF
C                    MOVE 'P'       QTYPE                 P=Print
```

```
C                      ELSE
C                      MOVE 'D'      QTYPE              D=Display
C                      END
C                      END
C*
C           1          DO   5        FILLVL  10         Do max 5 formats
C                      MOVE FIL,FILLVLFILEDS             Load file DS
C           HEXZZ      COMP HEXFF                     33  If query defined
C  N33                 EXCPTFILOUT                       Output record
C  N33                 END                              End DO
C*
C                      ENDSR
O*****************************************************************
OQRYEXT  E                 FILOUT
O                          FILNAM    8
O                          DICNAM   16
O                          IFILNM   24
O                          IFMTNM   32
O                          QRYDSC   76
O                          LSTUSR   84
O                          FILLVL   85
O                          DRNAME   93
O                          DRDATE   99
O                          LIBNAM  107
O                          QTYPE   108
```

Figure 7-16

Procedure QRYXRF

```
*
* Procedure:  QRYXRF
* Parameters:  1-8 libraries containing Query modules.
*
// IF JOBQ-YES GOTO QUEUED
*
// TAG PROMPT
// PROMPT MEMBER-QRYXRFFM,FORMAT-PARAMTRS,LENGTH-'8,8,8,8,8,8,8,8,50'
// EVALUATE P41= P42= P43= P44= P45= P46= P47= P48= P09=
// IF ?CD?/2003 RETURN
// IF ?CD?/2007 CANCEL
*
// TAG EDIT
// IF ?1?/ GOTO P01END
// IF DATAF1-?1? GOTO P01END
// EVALUATE P09='LIBRARY NOT ON DISK' P41='X'
// GOTO PROMPT
*
// TAG P01END
// IF ?2?/ GOTO P02END
// IF DATAF1-?2? GOTO P02END
// EVALUATE P09='LIBRARY NOT ON DISK' P42='X'
// GOTO PROMPT
*
// TAG P02END
// IF ?3?/ GOTO P03END
// IF DATAF1-?3? GOTO P03END
// EVALUATE P09='LIBRARY NOT ON DISK' P43='X'
// GOTO PROMPT
*
// TAG P03END
// IF ?4?/ GOTO P04END
// IF DATAF1-?4? GOTO P04END
// EVALUATE P09='LIBRARY NOT ON DISK' P44='X'
// GOTO PROMPT
*
// TAG P04END
// IF ?5?/ GOTO P05END
// IF DATAF1-?5? GOTO P05END
// EVALUATE P09='LIBRARY NOT ON DISK' P45='X'
// GOTO PROMPT
*
// TAG P05END
// IF ?6?/ GOTO P06END
// IF DATAF1-?6? GOTO P06END
// EVALUATE P09='LIBRARY NOT ON DISK' P46='X'
// GOTO PROMPT
```

```
*
// TAG P06END
// IF ?7?/ GOTO P07END
// IF DATAF1-?7? GOTO P07END
// EVALUATE P09='LIBRARY NOT ON DISK' P47='X'
// GOTO PROMPT
*
// TAG P07END
// IF ?8?/ GOTO P08END
// IF DATAF1-?8? GOTO P08END
// EVALUATE P09='LIBRARY NOT ON DISK' P48='X'
// GOTO PROMPT
*
// TAG P08END
// IF ?CD?/2004 JOBQ ?CLIB?,QRYXRF,?1?,?2?,?3?,?4?,?5?,?6?,?7?,?8?
// IF ?CD?/2004 RETURN
*
// TAG QUEUED
// LOCAL OFFSET-1,DATA-'?1?',BLANK-84
// LOCAL OFFSET-9,DATA-'?2?'
// LOCAL OFFSET-17,DATA-'?3?'
// LOCAL OFFSET-25,DATA-'?4?'
// LOCAL OFFSET-33,DATA-'?5?'
// LOCAL OFFSET-41,DATA-'?6?'
// LOCAL OFFSET-49,DATA-'?7?'
// LOCAL OFFSET-57,DATA-'?8?'
*
// IF DATAF1-#QRYEXT DELETE #QRYEXT,F1
*
// SWITCH 0XXXXXXX
// LOAD QRYXRF
// FILE NAME-QRYEXT,LABEL-#QRYEXT,RECORDS-100,EXTEND-100
// RUN
*
// IF SWITCH1-1 PAUSE 'Terminal error occurred during run'
// IF SWITCH1-1 RETURN
*
// IDDULINK LINK,#QRYEXT,foldername,QRYEXT
```

Figure 7-17

IDDU specifications for file #QRYEXT (Part 1)

```
                    SELECT AND SEQUENCE FIELDS FOR A FORMAT
Definition        QRYEXT                   Dictionary          CARIBOU
Type choices, press Enter
 ITEM                             CHOICE    POSSIBLE CHOICES
 Field definition name                      Name, ALL to create multiple fields

 Sequence number                  120       0-9999
 Position list to                           Name or sequence number

                         LIST OF FIELD DEFINITIONS                   TOP
Or type sequence number(s), press Enter
SEQ   NAME     BEGIN LENGTH  DATA  COMMENT
  10  FILNAM      1     8    CHAR  File name
  20  DICNAM      9     8    CHAR  IDDU dictionary name
  30  IFILNM     17     8    CHAR  IDDU file name
  40  IFMTNM     25     8    CHAR  IDDU format name
  50  QRYDSC     33    44    CHAR  Query description
  60  LSTUSR     77     8    CHAR  Last user to maintain
  70  FILLVL     85     1 0  ZONE  File level
  80  QRYNAM     86     8    CHAR  Query name
  90  LSTCHG     94     6,0  ZONE  Date last changed
Cmd4-Show names only   Cmd12-Renumber        Roll-Page
Cmd3-Go back           Cmd5-Create field     Cmd8-Reset selections
```

Figure 7-18

IDDU specifications for file #QRYEXT (Part 2)

```
                     SELECT AND SEQUENCE FIELDS FOR A FORMAT
Definition        QRYEXT                   Dictionary:          CARIBOU
Type choices, press Enter.
 ITEM                          CHOICE   POSSIBLE CHOICES
 Field definition name                  Name, ALL to create multiple fields

 Sequence number . . . .       120      0-9999
 Position list to.                      Name or sequence number

                          LIST OF FIELD DEFINITIONS                  MORE
Or type sequence number(s), press Enter
SEQ  NAME     BEGIN LENGTH  DATA  COMMENT
 100  LIBNAM    100     8    CHAR  Query library name
 110  OUTYPE    108     1    CHAR  Output device (printer, display, or file)




Cmd4-Show names only   Cmd12-Renumber       Roll-Page
Cmd3-Go back           Cmd5-Create field    Cmd8-Reset selections
```

Figure 7-19

F- and I-specs for file #QRYEXT. (This is member #QRYEXT on diskette.)

```
*.      1    ... 2 .. ... 3       4        5         6 . . ... 7  .     8        9 .
     FQRYEXT  IP  F 108 108        DISK
     IQRYEXT  NS
     I                                         1   8 FILNAM                  Disk file name
     I                                         9  16 DICNAM                  IDDU dictionary
     I                                        17  24 IFILNM                  IDDU file name
     I                                        25  32 IFMTNM                  IDDU format name
     I                                        33  76 QRYDSC                  Query desc
     I                                        77  84 LSTUSR                  Last user update
     I                                        85  850FILLVL                  File level (1-5)
     I                                        86  93 QRYNAM                  Query name
     I                                        94  990LSTCHG                  Last maint date
     I                                       100 107 LIBNAM                  Query lib name
     I                                       108 108 QTYPE                   Query type
```

Figure 7-20

Sample report (partial) created from file #QRYEXT

```
07/05/90  12.09 52                           Query Format Cross Reference            (QRYFMTXR)         PAGE    1
Dictionary  IDDU     IDDU     File     Query    Query    Query                           File  Last     Date    Output
 Name       Format   File     Name     Name     Library  Description                     Level User     Last    Type
            Name     Name                       Name                                                    Changed
 @MAPICS1   ARCMTT00 ARCMTT   M ARCMTT ARCOMENT JODY     A/R Inquiry Comments Report    ARIO   1   OHWOWMAN 90/04/04  D
 @MAPICS1   CHECKB00 CHECKB   M.CHECKB CHECKSAP QUERIES  A/P Checks Inquiry (MAPICS # vs Bank #)  M  2  SCHRODER 90/03/26  D
 @MAPICS1   CHECKR00 CHECKR   M.CHECKR CHECKSPR QUERIES  P/R Checks Inquiry (MAPICS # vs Bank #)  M  2  SCHRODER 90/03/28  D
 @MAPICS1   CORHST01 CORHST   M CORHST CUSPOINO QUERIES  Look Up Order/Invoice from Customer PO  M  2  OHWOWMAN 90/03/16  D
 @MAPICS1   CORHST02 CORHST   M CORHST CUSPOINO QUERIES  Look Up Order/Invoice from Customer PO  M  1  OHWOWMAN 90/03/16  D
 @MAPICS1   CUSMAS00 CUSMAS   M CUSMAS CUSNOPAR QUERIES  Customers That Won't Accept Partial Shipment  1  TRAPPER  90/03/21  P
 @MAPICS1            CUSMAS   M.CUSMAS CUSLOXUP QUERIES  Customer Number Look Up by Name  M  1  TRAPPER  90/03/20  P
 @MAPICS1            CUSMAS   M CUSMAS CUSTNM   JODY     Customers 60000000 - 69999999       1  OHWOWMAN 90/03/21  P
 @MAPICS1            CUSMAS   M CUSMAS CUSCRED  QUERIES  Customer Credit Info by Rep    M    1  SCHRODER 90/03/20  P
 @MAPICS1            CUSMAS   M.CUSMAS MTDCMEMO QUERIES  Monthly List of Cr Memos by Cr Memo Number M  2  CAPNKIRK 90/07/03  P
 @MAPICS1            CUSMAS   M CUSMAS EDITCCLS QUERIES  Edit Customer Master for Invalid Cust Cls  M  1  TRAPPER  90/03/21  P
 @MAPICS1            CUSMAS   M CUSMAS FORFILLY JODY     All Customers- City,State,Cr Limit L,Payment  1  OHWOWMAN 90/03/21  P
```

```
@MAPICS1        CUSMAS   M CUSMAS INVSLMAN JODY     Invoices by Salesman   (M MTHACT)   MTHEND2A  2  OHWOWMAN 90/05/22  P
@MAPICS1        CUSMAS   M CUSMAS INSHIP   QUERIES  Pick Lists in Shipping                     M  2  SCHRODER 90/05/28  P
@MAPICS1        CUSMAS   M CUSMAS CUSTTERX QUERIES  Customer Address List   Select Terr/Rep    M  1  SCHRODER 90/04/26  P
@MAPICS1        CUSMAS   M CUSMAS CUSTTERS QUERIES                                                2  SCHRODER 90/04/26  P
@MAPICS1        CUSMAS   M CUSMAS CUSTHOLD JODY     Customers on Hold                        APS  1  OHWOWMAN 90/04/04  P
@MAPICS1        CUSMAS   M CUSMAS CUSTTERR QUERIES  Customer Address List - Select Terr/Rep    M  1  SCHRODER 90/06/06  P
@MAPICS1        CUSMAS   M CUSMAS CUSTREPA QUERIES  Active Customers by Rep                       1  TRAPPER  90/03/21  P
@MAPICS1        CUSMAS   M.CUSMAS CUSTOPSB QUERIES  Top Customers w/Sales > 5000 in Prev 12 Mths  2  SCHRODER 90/04/04  P
@MAPICS1        CUSMAS   M CUSMAS CUSTOPS  QUERIES  Top Customers w/Sales > 5000 in Prev 12 Mths  2  TRAPPER  90/03/21  P
```

Figure 7-21

Sample report field selections

```
                         SELECT AND SEQUENCE FIELDS                    ALL
      Query  QRYFMTXR         Library  QRY              Option  REVISE
Press Enter to confirm
Select the fields to appear in the report and specify the sequence by typing
numbers beside the names, or press Cmd11 to select all fields, press Enter

    SEQUENCE   NAME
       10      DICNAM
       20      IFMTNM
       30      IFILNM
       40      FILNAM
       50      QRYNAM
       60      LIBNAM
       70      QRYDSC
       80      FILLVL
       90      LSTUSR
      100      LSTCHG
      110      QTYPE

Cmd3-Go back     Cmd4-Show comments     Cmd5-Show report    Cmd6-Fast roll
Cmd7-End         Cmd10-Show files       Cmd11-Select all    Cmd12-Renumber
Cmd13-Show report layout                Roll-Page
```

Figure 7-22

Sample report sort sequence

```
SELECT SORT FIELDS                        ALL
      Query  QRYFMTXR         Library  QRY             Option  REVISE
 Press Enter to confirm
 Select up to 5 fields on which to sort records by specifying sort
 priority (1-5) and the order  A (ascending) or D (descending)  Press Enter

 SORT                           SORT
 PRTY A/D  NAME                 PRTY A/D  NAME
  1     A  DICNAM                         QTYPE
  2     A  IFMTNM
  3     A  IFILNM
           FILNAM
           QRYNAM
           LIBNAM
           QRYDSC
           FILLVL
           LSTUSR
           LSTCHG

 Cmd3-Go back     Cmd4-Show comments      Cmd5-Show report    Cmd6-Fast roll
 Cmd7-End         Cmd10-Show files        Cmd12-Renumber
 Cmd13-Show report layout                 Roll-Page
```

Re-creating Subroutine SUBRLR

If you don't have assembler subroutine SUBRLR, you can re-create it with procedure MKSUBRLR (you don't need IBM's Assembler Language Program Product to install SUBRLR). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRLR. You need to run MKSUBRLR only once because SUBRLR is subsequently linked into program QRYXREF when it is compiled.

```
// * 'Re-creating R-module SUBRLR in library #RPGLIB'
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000167'     Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9D3D94040000000400000000000C0000400000000991000520000001490'
// LOCAL OFFSET-273,DATA-+
'0409151531000000801386000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-28,EXTEND-28
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRLR member to insert object code
// LOAD $FEFIX
// RUN
HDR  3B8C SUBRL00000
PTF  92CC RSUBRLR,99, #RPGLIB
DATA 0E22 00 0000 E2D8E2E4C2D9D3D9000000000000000000000000000000000000000000000000
DATA 96E7 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 8463 00 0040 E3960035F2870FE2E4C2D9D3D940F14BF14C40404040404010158340201 5F3408
DATA 54A8 00 0060 0183F4000A35010183F801D27D8000F78103F28 2AD350101837500347 4101915
DATA 9F09 00 0080 E3300089C105C2070164BC0707000AF020A0AF40104153CF10C118000A0C0BF2
DATA 0811 00 00A0 81763601016375010BC2020111BC3635010C9C0708003501000020382 11A1505
DATA 7FBA 00 00C0 E3310088C163750108800000008C0300F0188F40104013CF20C114001180008F2
DATA 8848 00 00E0 81412C0201F2022C010C1418C0106170C0C00010C140031202820211A18C001
DATA 8716 00 0100 E33000C2C0C17F2020800010C170C14C70101007807 0F4C0112C0174F0112C0C8
DATA CF3F 00 0120 F401401C01UL1 9DF4E021 50C1 73CF00C1135010183003020782 2181 0000A0801
DATA 42F8 00 0140 E33000F0750108400000C110EC101830000E2877A30F00C11F3C1810001 0C17
DATA E250 00 0160 0C08F2011000C010C1700C0C000 10C140C17F2030800002C7A3824101814000807
DATA 00DC 00 0180 E3300172F0C1UL170C14C2010130400117 0C174F01120C08F4014010C10C1B0F4E
DATA E402 00 01A0 C2180C173802016080020 0380 10C1 38CFF00FF0E010C0002A231F1913000B0402
DATA 0658 00 01C0 E32F01 8E0C180C100F010C170C080F010C140C08F2010430F330113501018375
DATA 2758 00 01E0 C1084C30000C110E01C1830C0AC2010000C20200003 82022 1A180F000807030 1
DATA 92BC 00 0200 E3040163C0008700000000000000000000000000000000000000000000000000
DATA 1C5J 00 0220 0000000000000000000000000000000000000000000000000000000000000000
DATA 196A 00 0240 E30001 6F4000000000000000000000000000000000000000000000000000000000
DATA C338 00 0260 0000000000000000000000000000000000000000000000000000000000000000
DATA 677C 00 0280 E33A014B0BE7E7E7E7E7E7E7E7E73000000000000000000000000000000000000
DATA D0D5 00 02A0 0000000000000000000000000000000000000000000000000000000BE7F7E7F7 00
DATA 07A4 00 02C0 E33A011E8E7E7F7E7300000000000000000000000000000000000000000000000
DATA FFC8 00 02E0 00000000000000000000000000000000000000000C8C0000000000004000000
DATA 9888 00 0300 E31501FC410040800204000000000000000000000100000000000000000000
DATA 2471 00 0320 0000000000000000000000000000000000000000000000000000000000000005
DATA 7891 00 0340 E30C0C100000000001000000AC00F013000000000000000000000000000000000
DATA 2181 00 0360 0000000000000000000000000000000000000000000000000000000000000000
DATA 0500 00 0380 E33A0C64E2E4C2D9D3D940404 0C38697A8990B8788A340408 3D40F1F9F8F96B
DATA 9224 00 03A0 40F1F9F8F0C4D40C78199A840E3484007996143A9868840C1939340888987880 0
DATA 86F5 00 03C0 E30A0C5FA3A2409885A2859A48 5B400000000000000000000000000000000000
DATA 08C8 00 03E0 0000000000000000000000000000000000000000000000000000000000000000
DATA 4F52 00 0400 C000000F00000000000000000000000000000000000000000000000000000000
DATA 0828 00 0420 0000000000000000000000000000000000000000000000000000000000000000
DATA 3E88 00 0440 818C00000000000000000000000000000000000000000000000000000000000000
DATA E758 00 0460 0000000000000000000000000000000000000000000000000000000000000000
```

Continued

```
DATA C02E 00 0480 4040404040404040404040404040404040404040404040404040404040404040
DATA 0E01 00 04A0 4040404040404040404040404040404040404040404040404040404040404040
DATA 0ED7 00 04C0 4040404040404040404040404040404040404040404040404040404040404040
DATA 4CAF 00 04E0 4040404040404040404040404040404040404040404040404040404040404040
END  FD07
```

Documenting RPG Program LDA Usage

by Perry Gardai

program by Ted Holt

Code on diskette:
Procedure MAPLDA
RPG programs MPLD1, MPLD2, MPLD3

Now you can locate LDA positions across all members within a library. Utility MAPLDA generates three reports: a map of the positions of the LDA, a listing of LDA use by field name, and a listing of LDA use by field starting position.

To ensure data and system integrity when you perform maintenance on unfamiliar software, you need to understand how the programmer used the local data area (LDA) throughout the application. Unhappily, the LDA is one of the most widely used but least documented features in S/36 applications. Although SSP does not provide any means of mapping the use of the LDA across all RPG members within a library, utility MAPLDA and its three short programs do. They give you a graphic display of LDA use by printing an X in the encumbered LDA positions. In addition to this map, utility MAPLDA provides two reports with more detailed information about where and how an application uses the LDA.

Getting Started

Procedure MAPLDA and programs MPLD1, MPLD2, and MPLD3 comprise utility MAPLDA. You initiate procedure MAPLDA (Figure 7-23) by keying in MAPLDA followed by the library name to be analyzed. If you omit the library name, the procedure prompts for it before continuing. The $MAINT routine starts the process by extracting all the source members from the target library and placing them in disk file WRK.?WS?.1. In preparation for program MPLD1 (Figure 7-24), the file name of the target library is loaded into positions 1 through 8 of the LDA.

Program MPLD1 uses file WRK.?WS?.1 to produce the map (Figure 7-25), printing an X in the appropriate location for each of the 512 LDA positions used within the library. In addition, program MPLD1 creates file WRK.?WS?.2, which contains one record for each field name referenced in the User Data Structure (UDS) section of the RPG I-specs. The data contained in each record includes the UDS field name, the RPG source member name referencing it, the starting and ending positions of the field, the decimal value of the field, and any comment included in the input specification.

After program MPLD1 terminates, procedure MAPLDA sorts file WRK.?WS?.2 in the first of two sort routines. The first sort routine — by

LDA field name within RPG member name sequence — results in output file WRK.?WS?.3. Program MPLD2 (Figure 7-26) uses file WRK.?WS?.3 to produce a report listing LDA use by field name (Figure 7-27). MPLD2 is a straightforward program that prints one line on the output report for each WRK.?WS?.3 record it reads.

The second sort of file WRK.?WS?.2 — by LDA field starting position within field ending position sequence — results in output file WRK.?WS?.4. This file is input for program MPLD3 (Figure 7-28), which produces a report of LDA use by field starting position (Figure 7-29). Like program MPLD2, program MPLD3 is a simple read/write print program.

Because you use the CONTINUE-YES parameter on the printer files for programs MPLD1 and MPLD2, and CONTINUE-NO for the printer file associated with program MPLD3, a single spool entry contains all three printouts. Although the sequential production of the reports at the end of the procedure is efficient in most cases, the single spool entry eliminates your ability to identify and control the individual listings on the spool file. If you want to build three separate spool file entries, you can omit the CONTINUE parameter or change it to -NO for programs MPLD1 and MPLD2.

Limitations

Utility MAPLDA's three reports give a good picture of what is going on in the LDA, but there are a few limitations. Utility MAPLDA analyzes LDA usage within RPG programs only. If the target library contains source programs in languages other than RPG, the resulting reports will not reflect the entire picture of LDA use within the library. In fact, in the unlikely event you have a non-RPG member with an I in position 6 and UDS in positions 18 through 20, you must modify the procedure to ensure that program MPLD1 processes only RPG members; otherwise, unpredictable results occur. You can modify the procedure easily by requiring program MPLD1 to check the submember type for RPG and to process only those records contained within RPG source members.

Another limitation arises because not all LDA use within a library may be in programs. A prime example is information from a prompt screen loaded into the LDA for further processing requirements within a procedure.

The third limitation relates to MRT programs, which frequently use IBM's SUBR21 routine instead of the UDS to read and write the LDA. Utility MAPLDA doesn't "see" calls to SUBR21 as LDA references. You could modify your programs that call SUBR21 to use the UDS in the last RLABL parameter for SUBR21, thus making the LDA usage visible to utility MAPLDA.

Because of these three limitations, the utility cannot give, in all circumstances, a full picture of LDA usage within an application library. Nevertheless, utility MAPLDA is a powerful automated tool you can use to improve your understanding of an application's architecture and design before you start to modify it.

Figure 7-23

Procedure MAPLDA

```
* MAPLDA - SHOW LDA USAGE IN RPG PROGRAMS FOR A LIBRARY
*
// IF ?1?/ * 'Enter name of library to be searched; leave blank to cancel '
// IF ?1R?/ RETURN
*
// LOAD $MAINT
// FILE NAME-WRK.?WS? 1,BLOCKS-20,EXTEND-5,RETAIN-J
// RUN
// COPY FROM-?1R?,TO-DISK,LIBRARY-S,FILE-WRK.?WS?.1,NAME-ALL,RECL-96
// END
*
// LOCAL OFFSET-1,DATA-'?1?                    Library name in LDA 1-8
*
// LOAD MPLD1
// FILE NAME-SOURCE,LABEL-WRK.?WS?.1,DBLOCK-12
// FILE NAME-DISK,LABEL-WRK.?WS?.2,BLOCKS-4,EXTEND-4,RETAIN-J
// PRINTER NAME-REPORT,CONTINUE-YES
// RUN
*
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-WRK.?WS?.2
// FILE NAME-OUTPUT,LABEL-WRK.?WS?.3,BLOCKS-4,EXTEND-4,RETAIN-J
// RUN
     HSORTR    14A       3   48
     FNC   1  14                     FIELD NAME / MEMBER NAME
     FDC  15  48                     REST OF RECORD
// END
*
// LOAD MPLD2
// FILE NAME-DISK,LABEL-WRK.?WS?.3,DBLOCK-12
// PRINTER NAME-REPORT,CONTINUE-YES
// RUN
*
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-WRK.?WS?.2
// FILE NAME-OUTPUT,LABEL-WRK.?WS?.4,BLOCKS-4,EXTEND-4,RETAIN-J
// RUN
     HSORTR     8A       3X  48
     FNC  15  22                     FROM POSITION / TO POSITION
     FDC   1  48                     ENTIRE RECORD
// END
*
// LOAD MPLD3
// FILE NAME-DISK,LABEL-WRK.?WS?.4,DBLOCK-12
// PRINTER NAME-REPORT,CONTINUE-NO
// RUN
*
```

Figure 7-24

Program MPLD1

```
*. .    1 ..  .. 2  .  . . 3 .     4  .     5 .    . 6         7 .      8
     H    P064                     B                                  MPLD1
     F*** THIS PROGRAM -
     F***        - PRINTS A MAP SHOWING LDA USAGE IN RPG MEMBERS OF A LIBRARY
     F***        - BUILDS A WORK FILE FOR OTHER REPORTS
     F*
     F*** INDICATORS
     F*      01 - RECORD ID, // COPY RECORD
     F*      02 - RECORD ID, START OF TABLE/ARRAY
     F*      03 - RECORD ID, COMPILER DIRECTIVE OR COMMENT
     F*      04 - RECORD ID, I SPEC WITH UDS IN 18-20
     F*      05 - RECORD ID, I SPEC WITH FIELD DEFINITION
     F*      06 - RECORD ID, CATCH-ALL
     F*
     F*      21 - ONE-TIME CALCS HAVE BEEN COMPLETED
     F*      31 - RECORD IS I SPEC WITHIN UDS
     F*      32 - RECORD IS MEMBER OF A TABLE/ARRAY
     F*
```

```
F*      51 - ERROR FOUND IN I SPEC
F*      52 - LOOP CONTROL, SUBROUTINE PROCES
F*      61 - LOOP CONTROL, SUBROUTINE CHART
F*
F*      81 - EXCPT LINE INDICATOR
F*      82 - EXCPT LINE INDICATOR
F*
FSOURCE  IP      96  96     2        DISK
FDISK    O       48  48     2        DISK
FREPORT  O      132 132    2OF       PRINTER
E*  Array LDA should be defined with 256 elements on S/34, 512 on S/36
E                     LDA       512  1                  CHART OF LDA USAGE
E                     LDA100    100  1                  SUBSTRING OF ARRAY LDA
ISOURCE  NS  01    1 C/   2 C/   3 C
I       AND        4 CC   5 CO   6 CP
I*                               // COPY STATEMENT
I                                      24  31 MEMBER                MEMBER NAME
I        NS  02    1 C*   2 C*   3 C
I*                               START OF TABLE/ARRAY
I        NS  03    7 C*
I       OR         7 C/
I*                               COMMENT OR COMPILER DIRECTIVE
I        NS  04    6 CI  18 CU  19 CD
I       AND       20 CS
I*                               UDS RECORD
I        NS  05    6 CI  15 C   19 C
I*                               I SPEC WITH FIELD DEFINITION
I                                      44  470FROM                  FIELD BEGINNING POS
I                                      48  510TO                    FIELD ENDING POS.
I                                      52  52 DEC                   DECIMAL PLACES
I                                      53  58 FIELD                 FIELD NAME
I                                      75  96 COMMEN                COMMENT/PROGRAM ID
I        NS  06
I*                               CATCH ALL
I          UDS
I                                       1   8 LIBR                  LIBRARY NAME
C* - BEGIN ONE-TIME CALCS                          HILOEQ
C  N21                MOVE '.'       LDA                FILL ARRAY LDA WITH DOTS (1ST TIME)
C*  Factor 2 of the following line should be 256 for S/34, 512 for S/36
C  N21                Z-ADD512       LDALEN  40         DEFINE LDA LENGTH
C  N21                SETON                        21   DO NOT REPEAT 1-TIME CALCS
C* - END ONE TIME CALCS                            HILOEQ
C*
C   01                SETOF                        3132
C*
C   02                SETON                        32   BEGINNING OF TABLE/ARRAY
C*
C   04                SETON                        31   BEGINNING OF LDA
C*
C   05 32                                               SKIP COMPILE TIME TABLE/ARRAY DATA
COR 05N31             SETOF                        05   I SPEC IS NOT INSIDE LDA
C*
C   05                EXSR PROCES                       I SPEC INSIDE OF LDA
C*
C   06                SETOF                        31   INDICATE THAT WE'RE NOT IN LDA FIELDS
C**
CLR                   EXSR CHART
C***********                                      HILOEQ
C          PROCES     BEGSR                             PROCESS I SPEC WITHIN LDA
C*
C          FROM       COMP 0001                      51 FROM/TO MUST BE IN RANGE
C  N51     TO         COMP LDALEN                  51       1 TO LDALEN
C   51                GOTO PROC90
C*
C                     Z-ADDFROM      X        40
C*
C          PROC50     TAG
C*
C                     MOVE 'X'       LDA,X              PUT X'S IN LDA ARRAY IN .
C                     ADD  0001      X                  .   POSITIONS BETWEEN FROM AND TO
```

```
C           X         COMP TO                    52
C  N52                GOTO PROC50
C*
C           PROC90    TAG
C                     ENDSR
C***********                                     HILOEQ
C           CHART     BEGSR                            PRINT CHART OF LDA USAGE
C*
C                     SETON                      81
C                     EXCPT                            PRINT SCALE
C                     SETOF                      81
C*
C                     Z-ADD0001      F        40
C*
C                     SETON                      82    SET TO PRINT EXCPT LINE
C*
C           CHAR20    TAG
C*                                               HILOEQ
C           F         ADD  0099      T        40
C           T         COMP LDALEN                61
C   61                MOVE LDALEN    T
C                     MOVE *BLANKS   LDA100
C                     MOVEALDA,F     LDA100
C                     EXCPT                            PRINT PART OF LDA
C  N61                ADD  0100      F
C  N61                GOTO CHAR20
C*
C                     SETOF                      82    DON'T PRINT EXCPT LINE ANYMORE
C*
C                     SETON                      81
C                     EXCPT                            PRINT SCALE
C                     SETOF                      81
C*
C                     ENDSR
ODISK    D        05N51
O                         FIELD      6
O                         MEMBER    14
O                         FROM      18
O                         TO        22
O                         DEC       23
O                         COMMEN    45
OREPORT  H  305   1P
O        OR       OF
O                         UDATE Y    8
O                                   76 'LDA USAGE FOR LIBRARY '
O                         LIBR      84
O                                  120 'PAGE'
O                         PAGE     124
O        D  1     05 51
O                                    8 '*ERROR*'
O                         MEMBER    20
O                         FIELD     28
O                         FROM  3   34
O                         TO    3   40
O        E  2     81
O                                   34 '....+....1....+....2'
O                                   54 '....+.   3  ..+....4'
O                                   74 '....+..  5   .+.. .6'
O                                   94 ' ...+.. .7.. .+....8'
O                                  114 '....+....9.   +  ..0'
O        E  2     82
O                         F     3    6
O                                    8 '-'
O                         T     3   12
O                         LDA100   114
```

Figure 7-25

Map of LDA use

```
1/18/89                                 LDA USAGE FOR LIBRARY NEWS3438                                    PAGE   1
                 .+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0
  1 - 100   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
101   200   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
201 - 300   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX....................................XX
301 - 400   XXXXXXXXXXXXXXXXXXXXX............................................................................
401 - 500      ................................................................................................X
501 - 512   XXXXXXX..
                 .+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0
```

Figure 7-26

Program MPLD2

```
*..      1 ... ... 2 .   ... 3  .. .   4   .     5          6     .    7        . 8
     H     P064                      B                                        MPLD2
     F*** PRINT REPORT OF LDA USAGE IN RPG PROGRAMS BY FIELD NAME
     F***
     FDISK    IP     48  48     2       DISK
     FREPORT  O     132 132     20F     PRINTER          .
     IDISK    NS  01
     I                                        1   6 FIELD L1
     I                                        7  14 MEMBER
     I                                       15  180FROM
     I                                       19  220TO
     I                                       23  23 DEC
     I                                       24  45 COMMEN
     I           UDS
     I                                        1   8 LIBR                     LIBRARY NAME
     OREPORT  H  205     1P
     O        OR         OF
     O                          UDATE Y    8
     O                                    72 'LDA USAGE FOR LIBRARY '
     O                          LIBR      80
     O                                   120 'PAGE'
     O                          PAGE     124
     O        H  2       1P
     O        OR         OF
     O                                    72 'BY FIELD NAME'
     O        H  1       1P
     O        OR         OF
     O                                     9 'FIELD'
     O                                    18 'MEMBER'
     O                                    26 'FROM'
     O                                    32 'TO'
     O                                    42 'DECIMALS'
     O                                    54 'COMMENT/ID'
     O        H  1       1P
     O        OR         OF
     O                                     9 '--- '
     O                                    18 '----'
     O                                    26 '---'
     O                                    32 '-'
     O                                    42 '-----'
     O                                    66 '---------------'
     O        D  1       01
     O                      L1  FIELD     10
     O                          MEMBER    20
     O                          FROM  3   26
     O                          TO    3   32
     O                          DEC       38
     O                          COMMEN    66
     O        T  1       L1
```

Figure 7-27

List of LDA use by field name

```
1/18/89                             LDA USAGE FOR LIBRARY NEWS3438
BY FIELD NAME
FIELD   MEMBER    FROM     TO  DECIMALS  COMMENT/ID

CHAR1   MASQCH       7     12            MASQCH

CHAR2   MASQCH      19     24            MASQCH

CHAR3   MASQCH      31     36            MASQCH

CODE    INV01       10     10     0

CONST1  NEWS02      67     86

CONST2  NEWS02      87    106

CONST3  NEWS02     107    126

CONST4  NEWS02     127    146

CONST5  NEWS02     147    166

CONST6  NEWS02     167    186

COUNT   ADDDUP     100    104     0      ADDDUP
CONDUP     100   104     0     CONDUP
CONDU1     100   104     0     CONDU1
NDUP2      100   104     0     NDUP2
NDUP22     100   104     0     NDUP22
PHODUP     100   104     0     PHODUP
PHODU2     100   104     0     PHODU2
    .
```

Figure 7-28

Program MPLD3

```
* . .   1 . . . 2 . .    3 . .  . 4        5         6 . .  . . 7      . . 8
     H    P064                             B                             MPLD3
     F*** PRINT REPORT OF LDA USAGE IN RPG PROGRAMS BY FIELD STARTING POSITION
     F***
     FDISK    IP      48  48     2        DISK
     FREPORT  O      132 132     20F      PRINTER
     IDISK    NS  01
     I                                              1   6 FIELD
     I                                              7  14 MEMBER
     I                                             15  180FROM  L1
     I                                             19  220TO
     I                                             23  23 DEC
     I                                             24  45 COMMEN
     I            UDS
     I                                              1   8 LIBR                   LIBRARY NAME
     OREPORT  H   205     1P
     O        OR          OF
     O                             UDATE Y    8
     O                                       72 'LDA USAGE FOR LIBRARY '
     O                             LIBR      80
     O                                      120 'PAGE'
     O                             PAGE     124
     O        H   2       1P
     O        OR          OF
     O                                       77 'BY FIELD STARTING POSITI'
     O                                       79 'ON'
     O        H   1       1P
     O        OR          OF
     O                                        6 'FROM'
     O                                       12 'TO'
     O                                       20 'FIELD'
     O                                       30 'MEMBER'
     O                                       42 'DECIMALS'
     O                                       54 'COMMENT/ID'
     O        H   1       1P
     O        OR          OF
     O                                        6 '—'
```

```
0                                          12 '-'
0                                          20 '--'
0                                          30 '---'
0                                          42 '----'
0                                          66 '------------'
0         D  1      01
0                               FROM  3     6
0                               TO    3    12
0                               FIELD      21
0                               MEMBER     32
0                               DEC        38
0'                              COMMEN     66
0         T  1      L1
```

Figure 7-29

List of LDA use by field starting position

```
1/18/89                      LDA USAGE FOR LIBRARY NEWS3438
BY FIELD STARTING POSITION
FROM    TO   FIELD     MEMBER     DECIMALS  COMMENT/ID

1      1   YESNO1    NEWSO2
1      2   LASTM     DEFINC        0
1      2   LASTM     INCSCH        0
1      2   WSIDNO    NEWCNT
1      2   WSIDNO    TMPOIN
1      6   INVNO     INVO1         0
1      6   INVNO     INVO2         0
1      6   MASS1     MASQCH               MASQCH
1      8   LDAFIL    TESTU

2      2   YESNO2    NEWSO2

3      3   YESNO3    NEWSO2
3      4   LASTY     DEFINC        0
3      4   LASTY     INCSCH        0
3     10   WSUSER    NEWCNT
3     10   WSUSER    TMPOIN

4      4   YESNO4    NEWSO2

5      5   YESNO5    NEWSO2

6      6   YESNO6    NEWSO2

7     10   START1    NEWSO2        0
7     12   CHAR1     MASQCH               MASQCH

9      9   LDATYP    TESTU
```

Documenting RPG Structured Opcodes

by Perry Gardai

program by Bruce Stradling

Code on diskette:

Procedure DOGRP
RPG program DOGRP
Screen format member DOGRPFM

Little by little, the distinctions between RPG II and RPG III are beginning to diminish. Under Release 4.0 of the S/36 SSP, IBM has supplied the RPG II programmer with the structured programming opcodes (IFxx, DOUxx, DOWxx, and CASxx) so zealously touted by advocates of the S/38. These codes offer a dramatic reduction in indicator use. But because each opcode must terminate with an END opcode, and because an error caused by the absence of a required END statement can be difficult to detect, debugging a program that uses Do Group opcodes is often time-consuming.

Procedure DOGRP and its related program give you a tool that visually establishes the relationships between structured programming (Do Group) opcodes and END statements. The utility reads a source member and produces a listing that connects each structured programming opcode to its corresponding END statement with a set of vertical lines; as an added benefit, the utility indents nested Do Group operations. When the utility is used during the debugging process, a violation of the END statement requirement becomes evident when the dots emanating from the Do Group opcode do not match up with an END statement. In addition, the indentation of nested Do Groups and related operations helps the maintenance programmer follow the logic of the program.

Figure 7-30a

Prompt screen for DOGRP utility

```
                    RPG - Do Group Source Listing Module

               This module list and ties the Do RPG Operators
                 with corresponding Else and End Operators

Enter RPG Source Member Name to be listed                          ________

Enter Library Name containing RPG source member                    ________

List the entire source or just the "C" specifications?  (ALL,CONLY)  CONLY




Cmd3-Previous Menu  Cmd4-Place on JOBQ  Cmd5-Display Via Copyprt
```

The utility consists of a prompt screen (Figure 7-30a; Figure 7-30b is the prompt screen format member), procedure DOGRP (Figure 7-31), and program DOGRP (Figure 7-32).

Utility DOGRP is called with four parameters: parameter 1 is the name of the program you are debugging, parameter 2 is the name of the library in which it resides, and parameter 3 is the portion of the program you want processed. If the entire source member is to be listed, enter ALL for parameter 3. If only the C-specs need to be listed, enter CONLY.

In addition, an optional fourth parameter is available to those programmers who want to conserve paper or system resources; the parameter allows the programmer to display the DOGRP listing on the CRT. To use the optional parameter to view the DOGRP listing on your workstation, key ",CRT" following the value for the third parameter, or press Command key 5 from the prompt screen. To send the DOGRP job to the job queue, press Command key 4 from the prompt screen. This precludes using the CRT option with the JOBQ option.

Procedure DOGRP (Figure 7-30) begins by determining whether parameters 1 and 2 have values. If they are blank, procedure DOGRP forces the current library into parameter 2 via an EVALUATE statement. If at this point the listing is not directed to the job queue, procedure DOGRP displays prompt screen DOGRPFM, which requires the programmer to supply values for parameters 1 and 3 (and to change the value of parameter 2, if necessary).

The procedure then determines whether Command keys 3, 4, 5, or 7 were used. Command key 3 terminates the procedure and returns control to the previous menu. Command key 4 directs the remainder of the procedure to the job queue to free up the workstation, and Command key 5 directs the debugged listing to the user's workstation. Command key 7 cancels procedure DOGRP and returns control to the master procedure.

Procedure DOGRP then ensures that parameters 1 and 2 have established values. If either parameter is still blank, the procedure once again prompts for the values. This time the procedure asks for the information by using standard screen messages (e.g., Enter RPG Source Program to be Documented). If all of these efforts fail to establish the necessary values, the procedure is canceled, and control is returned to the previous menu or master procedure.

After all housekeeping functions are out of the way, the real work begins. The procedure first loads parameters 1, 2, and 3 into the LDA via three // LOCAL OFFSET statements. The values established in the LDA subsequently are passed to program DOGRP. Next, the procedure uses a $MAINT routine to copy the specified source member into disk file DOGROUP, which will be read by program DOGRP. After file DOGROUP is created, the procedure loads and executes program DOGRP (Figure 7-32).

This program is written using Do Group logic. In fact, the Do Group logic used within program DOGRP allows file DOGROUP to be read within one RPG cycle. Remember, each record of file DOGROUP represents one line of the original RPG II source member. For the purposes of this article, program DOGRP is also the program used as input for the debug listing. Therefore, Figure 7-32 is the result of running program DOGRP against its own source member. As you can see, this listing of program DOGRP lets you understand Do Group logic quite easily.

To determine when and how much to indent the Do Group logic in the selected source member, program DOGRP uses four fields (defined in its C-specs) to act as indexes to arrays that hold the starting column positions of each line to be printed. Array index I1 controls movement to the right of information in the output LNE array, which holds the indented version of the line from the original source member (in 120 one-byte elements). Array index I2 controls the placement of an ELSE statement, ensuring that it corresponds with its respective IF statements. Array index I3 is the index to array IDX, which stores the column numbers indicating how far a line has been moved to the right when it is indented. Index I4 controls the placement of a nonstructured programming opcode so that it prints two positions to the right of the Do Group in which it is nested.

In program DOGRP, the first Do Group loop (lines 43 through 58) does two things until a C-spec is read: 1) it branches to subroutine LOAD; and 2) by using exception output, all RPG specs that precede the C-specs in the selected source member are printed (i.e., if the user specified ALL to the third parameter). Within subroutine LOAD, heading information is developed by manipulating the // COPY statement (i.e., the first record in the file created by $MAINT utility) to pick up the source member's reference number, date, and time (lines 138 through 162).

When a record containing a C-spec is read, another Do Group loop (lines 59 through 128) determines whether the record contains a Do Group opcode and, if it does, indents the opcode (when necessary) and inserts the appropriate number of dots based on the type of operation and the amount of nesting. If the code is a Do Group code, the line is indented from column 28. The codes IF, DO, and the first CAS are indented two spaces, while the codes ELSE, END, and additional CAS are not indented. If the code is not a Do Group code, the line is indented two spaces and the record is printed.

As each C-spec is analyzed and the appropriate number of dots inserted, the entire line is moved to array LNE. This array is output, and the result is the indented style of the debug report with each Do Group opcode visually connected to its corresponding END. As a final step, program DOGRP again determines whether ALL was specified to the third parameter. If so, it loops through the remaining RPG specifications from the selected source member and prints them (lines 129 through 135). At Last Record (LR) time (i.e., when the LR indicator comes on), the program branches to the #TG002 tag (line 137) and ends.

To END or not to END. That is the question answered by procedure DOGRP. With this new debugging tool, you can let your eyes find the errors your logic is not always able to detect. I don't know about you, but to me, the visual picture drawn by DOGRP is worth more than a thousand words in an RPG II compile listing.

Figure 7-30b

Screen format member DOGRPFM

```
*        1         2         3         4         5         6 .   ..7 . .   8
SPROMPT    0124      YYY       Y                                  CDEG
D          36 119Y                Y                      CRPG - Do Group Source LX
Disting Module
D          79 4 1Y                                       C              This modulX
De list and ties the Do RPG Operators
D          79 5 1Y                                       C                with corX
Dresponding Else and End Operators
D          70 8 1Y                                       CEnter RPG Source MemberX
D Name to be listed                                .. . .
DPARM01      8 87201 Y         Y             Y
D          7010 1Y                                       CEnter Library Name contX
Daining RPG source member
DPARM02      8107202 Y         Y             Y
```

```
14 D         7012 1Y                                   CList the entire source X
15 Dor just the "C" specifications? . (ALL,CONLY) .
16 DPARM03      5127203 YA      Y                      CCONLY
17 D         1820 1Y                                   CCmd3-Previous Menu
18 D         182021Y                                   CCmd4-Place on JOBQ
19 D         242041Y                                   CCmd5-Display Via CopyprX
20 Dt
```

Figure 7-31

Procedure DOGRP

```
// IF ?2?- EVALUATE P2-?CLIB?
// IF JOBQ-YES GOTO JOBQ
// IF ?1?- PROMPT MEMBER-DOGRPFM,FORMAT-PROMPT,START-1,LENGTH-'8,8,3'
// IF ?CD?-2003 RETURN
// IF ?CD?-2004 JOBQ ?CLIB?,DOGRP,?1?,?2?,?3?
// IF ?CD?-2004 RETURN
// IF ?CD?-2005 EVALUATE P4-CRT
// IF ?CD?-2007 RETURN
// TAG JOBQ
// IF EVOKED-NO IF JOBQ-NO IF ?1?/ * 'ENTER RPG SOURCE PROGRAM TO BE DOCUMENTED'
// IF ?1R?/   RETURN
// IF EVOKED-NO IF JOBQ-NO IF ?2?/ * 'ENTER NAME OF LIBRARY CONTAINING SOURCE'
// IF ?2R?/   RETURN
// LOCAL OFFSET-1,BLANK-19,DATA-'?1?'
// LOCAL OFFSET-9,DATA-'?2?'
// LOCAL OFFSET-17,DATA-'?3?'
// LOAD $MAINT
// FILE NAME-DOGROUP,UNIT-F1,BLOCKS-200,RETAIN-J
// RUN
// COPY FROM-?2?,TO-DISK,FILE-DOGROUP,RECL-120,NAME-?1?,LIBRARY-S,SVATTR-YES
// END
// LOAD DOGRP
// FILE NAME-INPUT,LABEL-DOGROUP
// IF ?4?-CRT PRINTER NAME-OUTPUT,PRIORITY-0,FORMSNO-DGRP
// RUN
// IFF ?4?/CRT RETURN
// LOAD $UASF
// RUN
// SPOOL SPOOLID-FDGRP,NAME-DOGRP?WS?,RELCANS-CANCEL
// END
// IF JOBQ-YES MSG ?WS?,DO GROUP LISTING IS IN FILE NAME-DOGRP?WS?
// IF JOBQ-YES RETURN
// LOAD $UASC
// FILE NAME-DOGRP?WS?,DISP-SHR
// RUN
// * 'DELETE FILE DOGRP?WS?,F1??? (NO-0 YES-1)'
// IF ?R?/1 DELETE DOGRP?WS?,F1
```

Figure 7-32

Program DOGRP (shown after run through utility DOGRP)

```
0001 H      64                       B       1                             DOGRP
0002 FINPUT   ID  F      120            DISK
0003 FOUTPUT  O   F      132     OF     PRINTER
0004 E                    LNE       105  1
0005 E                    IDX        20  2 0
0006 E                    CPY       120  1
0007 I*@*************************************************************
0008 I*@                   CALC.                      FLD  FILE  USAGE  OTHR
0009 I*@ IND. INPUT/OUTPUT COND. COMP. ARITH  LOKUP REC  NRF ERR EOF  USES
0010 I*@  LR  . . . . . .   C   . . . . . . . . . .     .   .   X    X
0011 I*@  OF    . . . . P   .   . . . . . . . . .           .        X
0012 I*@  60  . . . . . .   C   . . . . . . . . E       .   .        .
0013 I*@  70  . . . . . P   C   . . . . . . . . . . .   .   .   .    X
0014 I*@ INDICATOR USAGE SUMMARY INSERTED  3/03/86 AT 16.25.34 BY BRUCE
0015 I*@*************************************************************
0016 IINPUT   NS
0017 I                                  1    7 COPY                 A  7
0018 I                                  6    6 TYPE                 A  1
0019 I                                  7    7 ASTER                A  1
```

```
I                                       28  32 OP5                   A  5
I                                       28  29 OP2                   A  2
I                                       28  30 OP3                   A  3
I                                        1  27 REC27                 A 27
I                                       28 120 REC93                 A 93
I                                        1 120 CPY                   A120
I            DS
I                                        1  120TD                    N 12.0
I                                        1   40STIME                 N  4.0
I                                        7  120SDATE                 N  6.0
I                                       13  180REFNO                 N  6.0
I                                       19  200MM                    N  2.0
I                                       21  220DD                    N  2.0
I                                       23  240YY                    N  2.0
I                                       19  240DATE                  N  6.0
I                                       25  280TIME                  N  4.0
I           UDS
I                                        1   8 MEMBER                A  8
I                                        9  16 LIBR                  A  8
I                                       17  19 ALL                   A  3
C                     TIME           TD
C                     SUB  1         I1      20
C                     SETON                      OF
C           TYPE      DOUEQ'C'
C                       READ INPUT                    LR              INPUT   - READ
C   LR                  GOTO #TG002
C           COPY        CASEQ'// COPY' LOAD
C                       END
C           COPY        IFNE '// COPY'
C           COPY          IFNE '// CEND'
C           ALL             IFEQ 'ALL'
C           TYPE              IFNE 'C'
C                               EXCPTPRTALL
C  N70                          SETON                         70
C                             END
C                           END
C                         END
C                       END
C                     END
C           TYPE      DOWEQ'C'
C           ASTER       IFEQ '*'
C                         EXCPTPRTALL
C  N70                    SETON                    70
C                         GOTO #TG001
C                       END
C           OP2         IFNE 'IF'
C           OP2           IFNE 'DO'
C           OP3             IFNE 'CAS'
C           OP5               IFNE 'END  '
C           OP5                 IFNE 'ELSE '
C           I1                    ADD  2         I4      30
C                                 MOVEAREC93     LNE,I4
C                                 EXCPTPRTREC
C  N70                            SETON                     70
C                                 GOTO #TG001
C                               END
C                             END
C                           END
C                         END
C                       END
C           OP5         IFNE 'ELSE '
C           OP5           IFNE 'END  '
C           OP3             IFNE OP3SAV
C                             ADD  2         I1
C                           END
C                         END
C                       END
C           OP3         IFEQ 'CAS'
C                         MOVE OP3       OP3SAV  3
C                       ELSE
C                         MOVE '   '     OP3SAV
C                       END
C           OP2         IFEQ 'IF'
C                         ADD  1         I3      30
C                         Z-ADDI1        IDX,I3
C                       END
```

```
C           OP5         IFNE 'ELSE '
C                     . . MOVEAREC93     LNE,I1
C                     . ELSE
C                         Z-ADDIDX,I3    I2      30
C                       . MOVEAREC93     LNE,I2
C                       END
C                     . EXCPTPRTREC
C           OP5       . IFEQ 'ELSE '
C                     . . SUB  1         I3
C                     . END
C           OP5       . IFEQ 'END  '
C                     .   SUB  2         I1
C           I3          . IFNE 0
C           I1            . IFLT IDX,I3
C                     . . . . SUB  1         I3
C                     . . . END
C                         END
C                     . ELSE
C                     .   MOVEA'. '      LNE,I1
C                       END
C           #TG001    . TAG                                            #TG001- TAG
C                     . READ INPUT                    LR             INPUT    - READ
C   LR                . GOTO #TG002
C                     END
C           ALL       IFEQ 'ALL'
C           COPY      . DOUEQ'// CEND'
C                     .   EXCPTPRTALL
C                         READ INPUT                    LR             INPUT    - READ
C   LR                  . GOTO #TG002
C                       END
C                     END
C                     SETON                     LR
C           #TG002    TAG                                            #TG002- TAG
C           LOAD      BEGSR                                          LOAD  - SUBROUTINE
C                     Z-ADD1         L       30
C           #TG003    TAG                                            #TG003- TAG
C           'R'       LOKUPCPY,L                    60
C  N60                GOTO #TG004
C                     ADD  1         L
C           'E'       LOKUPCPY,L                    60
C  N60                GOTO #TG003
C                     ADD  1         L
C           'F'       LOKUPCPY,L                    60
C  N60                GOTO #TG003
C                     ADD  1         L
C           '-'       LOKUPCPY,L                    60
C  N60                GOTO #TG003
C                     ADD  1         L
C                     MOVEACPY,L     REFNO
C                     ADD  12        L
C                     MOVEACPY,L     YY
C                     ADD  3         L
C                     MOVEACPY,L     MM
C                     ADD  3         L
C                     MOVEACPY,L     DD
C                     ADD  8         L
C                     MOVEACPY,L     TIME
C           #TG004    ENDSR                                          #TG004- TAG
OOUTPUT  H   0305 OF
O                                    7 'LIBRARY'
O                                    9 '-'
O                         LIBR      18
O                                   33 'DATE'
O                         SDATE Y   42
O                                   52 'TIME'
O                         STIME     58 '  .  '
O                                   75 'PAGE'
O                         PAGE  Z   80
O                                  120 'DO GROUP LISTING'
O        H         OFN70
O                                   33 'REF'
O                                   42 'RECORD'
O        H         OFN70
O                                    6 'NAME'
O                                   20 'DATE'
O                                   27 'TIME'
```

```
0         35 'NUMBER'
0   H      0FN70
0          8 '-----'
0         22 '-----'
0         28 '---'
0         35 '-----'
0   H   10 0FN70
0          MEMBER    8
0          DATE  Y  22
0          TIME     28 '     '
0          REFNO    35
0   EF     PRTREC
0          REC27    27
0          LNE     132
0   EF     PRTALL
0          CPY     120
```

Detecting Duplicate or Outdated Members in Two Libraries

by Perry Gardai

program by Brian Blume

Code on diskette:
Procedure UTLLIB
RPG programs UTLIB1, UTLIB2, UTLIB3
Screen format member UTLLIBFM

Use utility UTLLIB to track and remove duplicate procedures and programs.

In nearly every data processing shop, changes to production applications take place in test libraries. Keeping track of which programs and procedures are current, which ones have been moved from the test library to the production library, and minimizing program and procedure redundancy between libraries can be an extremely laborious chore. Redundancy also can result when two production libraries with minimal program and procedure differences are set up to support two separate operation environments.

You should track and remove duplicate procedures and programs for several reasons. First, you can save significant disk space by minimizing program and procedure redundancy. Second, you can avoid inadvertent application errors by ensuring that test programs are not resident in production libraries. And third, you can ensure that only the newest version of a particular program is staged for production runs.

Utility UTLLIB, which detects duplicate and outdated programs and procedures between two libraries, consists of prompt screen UTLLIB (Figure 7-33); procedure UTLLIB (Figure 7-34); RPG programs UTLIB1, UTLIB2, and UTLIB3 (Figures 7-35 through 7-37, respectively); and screen format member UTLLIBFM (Figure 7-38). Utility UTLLIB compares the directory of one library — the source library — to the contents of a second library — the target library — by using the selected member name, member subtype, and date and time that member was last logged into a library. The result is a detailed exception report of any unmatched conditions existing between the directories of the two libraries.

Figure 7-33
UTLLIB prompt screen

```
                    S Y S T E M   3 6

                Library Comparison Utility

Source Library --->

Target Library --->

Output Report Info:

       Printer Id --->    P1           (Default - P1)

           Copies --->    01           (Default - 01)


Enter  Process    Cmd 7: Exit
                                                      UTLLIB
```

Procedure UTLLIB

Keying in UTLLIB activates procedure UTLLIB, which first performs some housekeeping to ensure that UTLLIB is not already active, that there is sufficient disk space to run the procedure, and to clear positions 256 through 275 of the LDA. Then, the prompt screen requires you to supply the names of the source and target libraries, the default setting of the printer ID, and the number of copies to be printed. The prompt screen is edited for errors and redisplayed with appropriate messages if errors are detected. When no errors are present, the remainder of the procedure is sent to the job queue so the terminal can be released for other functions.

When procedure UTLLIB starts to run from the job queue, all work files from previous runs of procedure UTLLIB are deleted, and program UTLIB1 is executed to create dummy file #DUMMY. The dummy file is copied to create a dummy library source member, #DUMMY, which is placed in both the source and target libraries via two TOLIBR commands because the LISTLIBR procedures that follow issue a terminal error if executed on "empty" libraries.

Next, procedures LISTLIBR and COPYDATA create disk files of the library directories of the source and target libraries. File #FILE01B, created as a sequential file from the source library directory, and file #FILE02B, created as an indexed file from the target library, are used as input to program UTLIB2. Program UTLIB2 is the core of this entire process.

How UTLLIB Works

Program UTLIB2 reads the records from file #FILE01B and, using member name and type as the key, chains to file #FILE02B. If the chain fails, the record (library directory entry) in the source library is for a new member. If the chain is successful — meaning the members exist in both libraries — the date and time information is compared. If the source date is earlier, it has not been updated by the target; if the source date is later, the target has not been updated; if the dates are the same, both versions are the same. Each time a chain fails or a date and time discrepancy exists, a consolidated record is written to #REPORT, the output file. Each record contains the source library name, member name, type and subtype, date and time logged into the directory, and the number of statements along with codes that designate each record's status: N for new member, U for updated member, or O for old member. The codes are used in the print program UTLIB3 to print the member status information.

File #REPORT now contains one record for each discrepancy found between the source and target libraries. If no records exist in file #REPORT, meaning the two libraries are identical, procedure UTLLIB sets switch one on and branches to the // TAG NOREC statement. Program UTLIB3 prints a report with the message ** No Members Found **. If exception records do exist, file #REPORT is renamed to #FILE and sorted back into file #REPORT to arrange the records in member name within member type sequence.

Program UTLIB3 now uses file #REPORT to produce the final listing (Figure 7-39). The report follows the basic format of a system-provided directory listing; the additions are the names of the source and target libraries being analyzed in the headings and the Status column. The Status column indicates the nature of the unmatched condition (NEW MEMBER, UPDATED VERSION, or OLDER VERSION) between the two libraries. After the report program is executed, all the work files are deleted, and the empty source member #DUMMY is removed from the source and target libraries.

You should note that procedure UTLLIB compares the directory of the source library to the target library directory; therefore, it cannot determine whether members exist in the target library that don't exist in the source library. To get this information, you have to run the procedure a second time with the library names reversed.

Although UTLLIB cannot actually provide library maintenance functions, it certainly can provide important information regarding the status of each member within the source library so the operator can determine the release level of each member in the working library.

Figure 7-34
Procedure UTLLIB

```
**
// IF JOBQ-YES  GOTO PROCESS               * PROCESS ON JOB QUEUE
**-------------------------------------
**       HOUSEKEEPING ROUTINE
**-------------------------------------
// IF ACTIVE-UTLLIB  GOTO ERRORD           * PROC 'UTLLIB' ACTIVE ALREADY
// IFF BLOCKS-100    GOTO ERRORE           * NOT ENOUGH SPACE AVAILABLE
// LOCAL OFFSET-256,BLANK-20               * BLANK OUT LDA
**-------------------------------------
**       MAIN PROMPT SCREEN
**-------------------------------------
// TAG RETRY
// EVALUATE P1-'?L'256,8'?'                * SOURCE LIBRARY LDA
// EVALUATE P3-'?L'264,8'?'                * TARGET LIBRARY LDA
```

```
// EVALUATE P5-'P1'                    * DEFAULT PRINTER ID
// EVALUATE P6-'01'                    * DEFAULT COPIES
// PROMPT MEMBER-UTLLIBFM,FORMAT-UTLLIB,START-1,LENGTH-'8,26,8,26,2,2'
// IF ?CD?/2007  RETURN
// LOCAL OFFSET-256,DATA-'?1?'         * ACCEPT INPUT SOURCE LIBRARY
// LOCAL OFFSET-264,DATA-'?2?'         * ACCEPT INPUT TARGET LIBRARY
// LOCAL OFFSET-272,DATA-'?3?'         * ACCEPT INPUT (P1 - 01)
// IF ?L'272,2'?/  LOCAL OFFSET-272,DATA-'P1'  * DEFAULT PARAMETER
// IF ?L'274,2'?/  LOCAL OFFSET-274,DATA-'01'  * DEFAULT PARAMETER
// EVALUATE P2-' '                     * BLANK OUT ERROR CODE
// EVALUATE P4-' '                     * BLANK OUT ERROR CODE
**_________________________________________
**      CHECK FOR ERRORS
**_________________________________________
// IF ?L'256,8'?/          GOTO ERRORB1
// IFF DATAF1-?L'256,8'?   GOTO ERRORC1
// IF ?L'264,8'?/          GOTO ERRORB2
// IFF DATAF1-?L'264,8'?   GOTO ERRORC2
**_________________________________________
**      GOOD DATA - PROCESS ON JOB QUEUE
**_________________________________________
// JOBQ 3,,UTLLIB
// RETURN
****************************************************************************
**      E R R O R   S U B R O U T I N E S
****************************************************************************
// TAG ERRORB1
// EVALUATE P2-'Library name is blank       '
// LOCAL OFFSET-256,BLANK-8
// GOTO RETRY
**
// TAG ERRORB2
// EVALUATE P4-'Library name is blank       '
// LOCAL OFFSET-264,BLANK-8
// GOTO RETRY
**
// TAG ERRORC1
// EVALUATE P2-'Source library not on disk'
// LOCAL OFFSET-256,BLANK-8
// GOTO RETRY
**
// TAG ERRORC2
// EVALUATE P4-'Target library not on disk'
// LOCAL OFFSET-264,BLANK-8
// GOTO RETRY
**
// TAG ERRORD
// PAUSE '*** ERROR  UTLLIB is already active   Job is canceled'
// RETURN
**
// TAG ERRORE
// PAUSE '*** ERROR  Not enough disk space to run UTLLIB   Job is canceled'
// RETURN
****************************************************************************
**      P R O C E S S   O N   J O B   Q U E U E
****************************************************************************
// TAG PROCESS
NOHALT 3,SESSION
**_________________________________________
**      DELETE SCRATCH FILES
**_________________________________________
// IF DATAF1-#DUMMY     DELETE #DUMMY,F1
// IF DATAF1-#FILE      DELETE #FILE,F1
// IF DATAF1-#FILE01A   DELETE #FILE01A,F1
// IF DATAF1-#FILE02A   DELETE #FILE02A,F1
// IF DATAF1-#FILE01B   DELETE #FILE01B,F1
// IF DATAF1-#FILE02B   DELETE #FILE02B,F1
// IF DATAF1-#REPORT    DELETE #REPORT,F1
**_________________________________________
**      COPY LIBRARIES TO DISK
**_________________________________________
// LOAD UTLIB1                         * BUILD DUMMY MEMBER FILE
// FILE NAME-OUTPUT,LABEL-#DUMMY,RECORDS-5
// RUN
**
```

```
TOLIBR #DUMMY,F1,,REPLACE,?L'256,8'?    * PLACE MEMBER INTO SOURCE LIBRARY
TOLIBR #DUMMY,F1,,REPLACE,?L'264,8'?    * PLACE MEMBER INTO TARGET LIBRARY
**      SOURCE LIBRARY
LISTLIBR DIR,LIBRARY,?L'256,8'?,....,#FILE01A
COPYDATA #FILE01A,,#FILE01B,.....,INCLUDE,24,EQ,'/'
**      TARGET LIBRARY
LISTLIBR DIR,LIBRARY,?L'264,8'?,....,#FILE02A
COPYDATA #FILE02A,,#FILE02B,.....,INCLUDE,24,EQ,'/',..,I,1,12
**________________________________________
**      BUILD REPORT FILE
**________________________________________
// LOAD UTLIB2
// FILE NAME-INPUT1,LABEL-#FILE01B,RETAIN-S
// FILE NAME-INPUT2,LABEL-#FILE02B,RETAIN-S
// FILE NAME-OUTPUT,LABEL-#REPORT,RECORDS-100,EXTEND-100
// RUN
**________________________________________
**      SORT REPORT FILE
**________________________________________
// IF ?F'A,#REPORT'?/00000000  SWITCH 1XXXXXXX
// IF ?F'A,#REPORT'?/00000000  GOTO NOREC
// RENAME #REPORT,#FILE
**
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-#FILE,RETAIN-S
// FILE NAME-OUTPUT,LABEL-#REPORT,DISP-NEW,RECORDS-?F'A,#FILE'?
// RUN
     HSORTR    11A       3X  40
     FNC  17  19                 * MEMBER TYPE
     FNC   9  16                 * MEMBER NAME
     FDC   1  40
// END
**________________________________________
**      PRINT REPORT
**________________________________________
// TAG NOREC
// LOAD UTLIB3
// FILE NAME-INPUT,LABEL-#REPORT,RETAIN-S
// PRINTER NAME-REPORT,DEVICE-?L'272,2'?,COPIES-?L'274,2'?
// RUN
**________________________________________
**      CLEAN UP ROUTINE
**________________________________________
// IF DATAF1-#DUMMY    DELETE #DUMMY,F1
// IF DATAF1-#FILE     DELETE #FILE,F1
// IF DATAF1-#FILE01A  DELETE #FILE01A,F1
// IF DATAF1-#FILE02A  DELETE #FILE02A,F1
// IF DATAF1-#FILE01B  DELETE #FILE01B,F1
// IF DATAF1-#FILE02B  DELETE #FILE02B,F1
// IF DATAF1-#REPORT   DELETE #REPORT,F1
**
// IF SOURCE-'#DUMMY,?L'256,8'?'  REMOVE #DUMMY,S,?L'256,8'?
// IF SOURCE-'#DUMMY,?L'264,8'?'  REMOVE #DUMMY,S,?L'264,8'?
```

Figure 7-35

Program UTLIB1

```
*    .  1 ... ... 2  .  .  3 .       4        5 .      6  .  .. 7 .   . . 8
0001 H     064                     B   1                               UTLIB1
0002 FOUTPUT  O    F  96  96           DISK
0003 C                    SETON                        LR
0004 OOUTPUT  T        LR
0005 O                                  7 '// COPY'
0006 O                                 30 'LIBRARY-S,NAME-#DUMMY'
0007 O        T        LR
0008 O                                  6 '// END'
```

Figure 7-36

Program UTLIB2

```
*   1         2         3         4         5         6         7         8
H     064                        B   1                                 UTLIB2
FINPUT1  IP  F 132 132            DISK
FINPUT2  IC  F 132 132R12AI     1 DISK
FOUTPUT  O   F  40  40            DISK                      A
IINPUT1  NS  01
I                                        1   8 MEM                   * MEMBER NAME
I                                       12  12 TYPE                  * MEMBER TYPE
I                                       16  18 SUBT                  * SUB TYPE
I                                       22  230MM                    * MONTH
I                                       25  260DD                    * DAY
I                                       28  290YY                    * YEAR
I                                       32  330HH                    * HOUR
I                                       35  360M                     * MINUTES
I                                      117 1190STMTS                 * # OF STATEMENTS
IINPUT2  NS  02
I                                        1   8 MEM2                  * MEMBER NAME
I                                       12  12 TYPE2                 * MEMBER TYPE
I                                       16  18 SUBT2                 * SUB TYPE
I                                       22  230MM2                   * MONTH
I                                       25  260DD2                   * DAY
I                                       28  290YY2                   * YEAR
I                                       32  330HH2                   * HOUR
I                                       35  360M2                    * MINUTES
I           DS                                                       *** DATA STRUCTURE
I* DEFINE REC01
I                                        1  22 REC01                 * RECORD ONE
I                                        1   8 MEM                   * MEMBER NAME
I                                        9   9 TYPE                  * MEMBER TYPE
I                                       10  12 SUBT                  * SUB TYPE
I                                       13  140YY                    * YEAR
I                                       15  160MM                    * MONTH
I                                       17  180DD                    * DAY
I                                       19  200HH                    * HOUR
I                                       21  220M                     * MINUTES
I* DEFINE REC02
I                                       25  46 REC02                 * RECORD TWO
I                                       25  32 MEM2                  * MEMBER NAME
I                                       33  33 TYPE2                 * MEMBER TYPE
I                                       34  36 SUBT2                 * SUB TYPE
I                                       37  380YY2                   * YEAR
I                                       39  400MM2                   * MONTH
I                                       41  420DD2                   * DAY
I                                       43  440HH2                   * HOUR
I                                       45  460M2                    * MINUTES
I           UDS                                                      *** LOCAL DATA AREA
I                                      256 263 SLIB                  * SOURCE LIBRARY
C*
C                     SETOF                     909192
C           MEM       COMP 'DUMMY                   99
C   99                GOTO BYPASS
C                     MOVELMEM       KEY     12
C                     MOVE TYPE      KEY
C           KEY       CHAININPUT2               90
C   90                EXCPTWRITE
C  N90      REC01     COMP REC02                9192
C   91
COR 92                EXCPTWRITE
C           BYPASS    TAG
C*
OOUTPUT  EADD             WRITE
O                         SLIB       8
O                         MEM       16
O                         TYPE      17
O                         SUBT      20
O                         MM        22
O                         DD        24
O                         YY        26
O                         HH        28
O                         M         30
O                         STMTS     33
```

```
0071 0                  90               40 'N'
0072 0                  91               40 'U'
0073 0                  92               40 'O'
```

Figure 7-37

Program UTLIB3

```
*... ... 1 ... .  2 ..  ... 3 ... ... 4  .. ... 5 . . ... 6  .. .   7 ... .. 8
0001 H     064                     B   1                             UTLIB3
0002 FINPUT   IP  F  40  40           DISK
0003 FREPORT  O   F 132 132     OF    PRINTER
0004 IINPUT   NS  01
0005 I                                      1   8 LIB   L2           * LIBRARY NAME
0006 I                                      9  16 MEM                * MEMBER NAME
0007 I                                     17  17 TYPE  L1           * MEMBER TYPE
0008 I                                     18  20 SUBT               * MEMBER SUB TYPE
0009 I                                     21  260DATE               * MEMBER DATE
0010 I                                     27  280HH                 * MEMBER HOURS
0011 I                                     29  300MM                 * MEMBER MINUTES
0012 I                                     31  330STMTS              * # OF STATEMENTS
0013 I*                                    34  39 FILLER             — OPEN SPACE
0014 I                                     40  40 CODE               NEW/UPDATED/OLD
0015 I            UDS                                                * LOCAL DATA AREA
0016 I                                    256 263 LIB1               * LIBRARY ONE SOURCE
0017 I                                    264 271 LIB2               * LIBRARY TWO TARGET
0018 C*
0019 C                      TIME           UTIME   60
0020 C            TOTAL     ADD  1         TOTAL   50
0021 C            CODE      COMP 'N'                      90
0022 C            CODE      COMP 'U'                      91
0023 C            CODE      COMP 'O'                      92
0024 C*
0025 OREPORT  D  104    L2
0026 0        OR        OF
0027 0                                 5 'DATE:'
0028 0                        UDATE Y  14
0029 0                                48 'S Y S T E M     3 6'
0030 0                                75 'PAGE:'
0031 0                        PAGE  Z  80
0032 0         D  2     L2
0033 0        OR        OF
0034 0                                 5 'TIME·'
0035 0                        UTIME   14 '  .     '
0036 0                                44 'LIBRARY COMPARISON'
0037 0                                52 'UTILITY'
0038 0                                76 'UTLLIB'
0039 0         D  2     L2
0040 0        OR        OF
0041 0                                20 'SOURCE LIBRARY ---> '
0042 0                        LIB1    30
0043 0                                43 'COMPARED TO'
0044 0                                65 'TARGET LIBRARY ---> '
0045 0                        LIB2    75
0046 0         D  1     L2
0047 0        OR        OF
0048 0                                 7 'LIBRARY'
0049 0                                17 'MEMBER'
0050 0                                27 'MEMBER'
0051 0                                36 ' SUB  '
0052 0                                45 'MEMBER'
0053 0                                56 'MEMBER'
0054 0                                64 '# OF '
0055 0                                85 'STATUS'
0056 0         D  1     L2
0057 0        OR        OF
0058 0                                 5 'NAME'
0059 0                                16 'NAME'
0060 0                                26 'TYPE'
0061 0                                35 'TYPE'
0062 0                                44 'DATE'
0063 0                                55 'TIME'
0064 0                                64 'STMTS'
```

```
O        D  2     L2
O       OR        OF
O                                   8 '________'
O                                  19 '________'
O                                  27 '______'
O                                  37 '________'
O                                  47 '________'
O                                  56 '______'
O                                  64 '____  '
O                                  90 '________________    '
O        D  1     L1
O        D  1     01
O                 L1       LIB      8
O                          MEM     19
O                          TYPE    25
O                          SUBT    35
O                          DATE  Y 47
O                          HH      53
O                                  54 ' '
O                          MM      56
O                          STMTS   64
O                 90               90 'NEW MEMBER'
O                 91               90 'UPDATED VERSION'
O                 92               90 'OLDER VERSION'
O        T  104   LR U1
O                                   5 'DATE '
O                          UDATE Y 14
O                                  48 'S Y S T E M    3 6'
O                                  75 'PAGE '
O                          PAGE  Z 80
O        T  2     LR U1
O                                  44 'LIBRARY COMPARISON'
O                                  52 'UTILITY'
O                                  76 'UTLLIB'
O        T  2     LR U1
O                                  20 'SOURCE LIBRARY ---> '
O                          LIB1    30
O                                  43 'COMPARED TO'
O                                  65 'TARGET LIBRARY ---> '
O                          LIB2    75
O        T 32     LR U1
O                                  50 '*** NO MEMBERS FOUND ***'
O        T 22     LRNU1
O                                  10 'TOTAL # OF'
O                                  27 'MEMBERS LISTED  '
O                          TOTAL 1 37
O        T 3      LR
O                                  17 '    END OF REPORT'
```

Figure 7-38

Screen format Member UTLLIBFM

```
*  .      1         2         3         4         5         6         7         8
SUTLLIB
D            17 430Y                    Y                   CS Y S T E M   3 6
D            26 625Y                    Y        Y          CLibrary Comparison UtilX
Dity
D            19 910Y                                        CSource Library --->
DLIB1         8 936Y  Y                            Y
D            26 950Y                      Y
D            191110Y                                        CTarget Library --->
DLIB2         81136Y  Y                            Y
D            261150Y                      Y
D            191310Y                                        COutput Report Info
D            151517Y                                        CPrinter Id --->
DPRINT        21536Y  Y                            Y
D            141550Y                                        C(Default - P1)
D            111721Y                                        CCopies --->
DCOPY         21736Y  Y                            Y
D            141750Y                                        C(Default - 01)
D            292210Y                                        CEnter  Process    Cmd 7X
D  Exit
D             62370Y                    Y                   CUTLLIB
```

Figure 7-39
UTLLIB detailed exception report

```
DATE:  12/16/66           S Y S T E M    36                       PAGE 1
TIME. 11:15:11        LIBRARY COMPARISON UTILITY

SOURCE LIBRARY ---> #UTILITY COMPARED TO TARGET LIBRARY ---> #LIBRARY

LIBRARY    MEMBER     MEMBER    SUB    MEMBER     MEMBER    # OF        STATUS
 NAME       NAME       TYPE     TYPE    DATE       TIME     STMTS
 ----      ----        ----     ----    ----       ----

#UTILITY   U08P          P              10/03/88  09:18     043         OLDER VERSION

#UTILITY   UMENU         S      MNU      2/22/88  09:00     048            NEW MEMBER
           UMENU##       S      MNU      2/22/88  09:00     022            NEW MEMBER

TOTAL # OF MEMBERS LISTED           3
```

Saving Print Screens as Source Members

by George A. Meyer

Code on diskette:
Procedure GEOPK
RPG program GEOPK

Utility GEOPK enables you to include screen images in a word processing file for documentation purposes. Procedure GEOPK (Figure 7-40) and RPG program GEOPK (Figure 7-41) capture screen images by loading output from the Print key into a library as source code. To copy a screen into a library, first stop the spool writer. Then call up the screen image that you want a hard copy of, and press the PRINT key. Hold the subsequent spool file on the print queue. While the spool writer is stopped, you can display, print via the PRINT key, and HOLD as many screen images as you want. Just keep a list of the screen images and their respective spool IDs.

Once you have all the screen images held on the spool file, run procedure GEOPK on each of the spool IDs. Procedure GEOPK has three parameters: the spool ID, the name of the library member that will contain the screen image, and a request for rerun or cancel. The procedure runs the COPYPRT procedure to copy the specified spool file into a disk file by the same name and saves the name of the file in the LDA. Then the procedure loads program GEOPK, which reads the file name from the LDA, writes a // COPY statement at the beginning of output file LIBO, copies the screen image file line by line to file LIBO, and writes a // CEND statement at end of file. Because the screen image file was not created by $MAINT, these OCL statements are needed by SSP procedure TOLIBR, which creates the library source member containing the screen image (for more information about // COPY and // CEND, see chapter 4 of the IBM *S/36 System Reference Manual*). Procedure TOLIBR is called by procedure GEOPK when program GEOPK has run. After you run procedure and program GEOPK, the screen image will be in the specified source member, ready for editing.

There are a few safety checks built into procedure GEOPK. If a file on disk has the same name as the one you specify in the spool ID parameter, or if a procedure, source, subroutine, or load member has the same name as the one you specify in the library member parameter, the procedure will be canceled. If you have any other potential name conflicts, you need to modify procedure GEOPK to include the appropriate checks.

We have found this procedure most useful for including problem screens in letters to software vendors and for local documentation. (We use the EDIT function of POP to add the additional text.)

Figure 7-40

Procedure GEOPK

```
// TAG RERUN
// EVALUATE P1=' '
// EVALUATE P2=' '
// EVALUATE P3=' '
*
// * ' ENTER SPOOL ID '
// IF DATAF1-?1R? GOTO A
// COPYPRT ?1?,?1?,CANCEL
*
// * ' ENTER NAME TO BUILD '
// LOCAL OFFSET-1,DATA-'?2R?'
*
// IF SOURCE-'?2?,?CLIB?  GOTO B
// IF SUBR-'?2?,?CLIB?'   GOTO B
// IF PROC-'?2?,?CLIB?'   GOTO B
// IF LOAD-'?2?,?CLIB?'   GOTO B
*
// LOAD GEOPK
// FILE NAME-PRINT,LAABEL-?1?
// FILE NAME-LIBO,LABEL-?2?,DISP-NEW,RECORDS-200,EXTEND-100
// RUN
*
// TOLIBR ?2? F1,,,?CLIB?,, ,ALL,LIBRARY
*
// IF DATAF1-?1? DELETE ?1?,F1,REMOVE
*
// TAGE C
// * ' ENTER Y TO RERUN OR N TO CANCEL '
// IF ?3R?-Y GOTO RERUN '
// IF ?3R?-N CANCEL
// ELSE      GOTO C
*
// TAG A
// PAUSE   ?1? FILE ALREADY EXISTS JOB IS CANCELED. PRESS 0 '
// CANCEL
*
// TAG B
// PAUSE '?2? LIBRARY MEMBERS EXIST JOB IS CANCELED. PRESS 0 '
// CANCEL
*
```

Figure 7-41

Program GEOPK

```
*        1         2         3         4         5         6         7         8
     H      24                                                           GEOPK
     FPRINT   IP  F 150 150            DISK
     FLIBO    O   F 96   96            DISK
     I*
     IPRINT   AA  01    1 CH
     I        OR  02    1NCH
     I                                         1    1 CH
     I                                        09   90 IP
     ILDA          UDS
     I                                         1    6 LN
     I*
     OLIBO    D         01NO2
     O        OR        02NO1
     O                  01NO2             23  // COPY LIBRARY-S,NAME-'
     O                  01NO2    LN       29
     O                  02NO1    IP       85
     OLIBO    T         LR                 7 '// CEND'
     O*
```

Files

CHAPTER

8

8

Accessing Files Dynamically from RPG

by Perry Gardai

program by Mel Beckman

Code on diskette:
Procedure FLEDIT
RPG program FLEDIT
Screen format member FLEDITFM
Assembler subroutine SUBRFA

Most computers and common high-level languages offer dynamic access to a file within a program, without compiling file attributes (e.g., record length) into the machine-executable version of the program. Such a feature, which IBM calls Special Allocate, is especially useful for text processors, file editors, communications file processing, and other applications for which the exact composition of a file is unknown. Special Allocate is an integral part of SSP used by many of IBM's own programs. Special Allocate's file access capabilities also can add power and flexibility to the RPG programmer's arsenal, but IBM unfortunately has not provided an interface between Special Allocate and RPG.

Such an interface is provided by subroutine SUBRFA. With SUBRFA, you can open numerous files simultaneously, without coding the // FILE statement in the program's calling procedure or defining the file in the RPG F-specs. You can open any type of file (sequential, indexed, or direct) with any record length and manipulate the file in any routine manner (e.g., add, change, and delete record; randomly access keyed or relative record numbers; read next and read prior). The only restriction is that the files must exist on disk; you can create new records, but not new files, on the fly.

Program FLEDIT (Figure 8-1) is one example of how SUBRFA is incorporated into a program. Although FLEDIT is a relatively unsophisticated file editor, it provides an *ad hoc* file edit capability absent in most S/36 installations. FLEDIT's simplicity makes it a good vehicle for becoming acquainted with SUBRFA. Program FLEDIT uses screen format member FLEDITFM (Figure 8-2) and is called by procedure FLEDIT (Figure 8-3).

Before you use program FLEDIT to call SUBRFA, you must understand the various RLABL (record label) statements SUBRFA requires to open, access, and close a file. Figures 8-4a through 8-4d offer you a detailed explanation of the RLABL code structures for each file function. The values supplied in these RLABLs control SUBRFA. After you code the RLABL statements, you can use SUBRFA with program FLEDIT, the two-screen file editor mentioned earlier.

The first screen of program FLEDIT asks for the name of the file to be opened, access type (I, O, or U), keyed file flag, and share level. Procedure

FLEDIT lets you skip this first screen by entering the call command in a format similar to

```
FLEDIT filename,mode,share, + keyflag
```

Parameter 1 is the name of the file to be opened. Parameter 2 must be U for update, I for input, or O for output, and defaults to U if unspecified. Parameter 3 is the share level (listed under the description of the PARMS RLABL in Figure 8-4c), which defaults to MM. A K for parameter 4 accesses the file through its keys; leave this parameter unspecified for unkeyed access.

After you enter the file name, screen two (Figure 8-5) displays records contained in the file. Initially, no record is displayed; you can press "roll up" to view the first record in the file. You then can manipulate the file by using the command and function keys displayed at the bottom of the screen.

The beauty of using Special Allocate via SUBRFA is that there is not a single // FILE statement in the procedure, yet program FLEDIT can edit any file on the system. FLEDIT uses only one file at a time, but you can employ the same principles to access any number of files within a single program. By comparing the manipulation of screen two and the code in FLEDIT, you quickly gain appreciation for the simplicity and power the Special Allocate function incorporates into an application program.

Figure 8-1

Program FLEDIT

```
*        1 .       2         3         4         5   .     6       . 7 .... 8
0001 H     064                     B       1                             FLEDIT
0002 F*
0003 F* Primitive file editor using SUBRFA
0003 F* By Mel Beckman
0004 F*
0005 FWORKSTN CD  F    2048            WORKSTN
0006 F                                              KINFDS INFDS
0007 E                    SEG        10  4 0                  Segment numbering
0008 E                    REC      4096  1                    Data record buffer
0009 E                    BIN      4096  1                    Binary data hold
0010 E                    R50        10 50                    Screen buffer
0011 E                    MSGKEY  1  22  6   MSG    60        Screen buffer
0012 I*
0013 I* Open file prompt screen input
0014 I*
0015 IWORKSTN          1 CO
0016 I                                        2   9 NAME
0017 I                                       10  13 PARMS
0018 I*
0019 I* File data screen input
0020 I*
0021 IWORKSTN          1 C1
0022 I                                        2 100 KEY
0023 I                                      101 1040NEWRP
0024 I                                      105 604 R50
0025 I*
0026 I* Data record data structure
0027 I*
0028 IRECORD      DS
0029 I                                        14096 REC
0030 I*
0031 I* File open feedback data structure
0032 I*
0033 IFEEDBK      DS
0034 I                                        1   80FFRUSD
0035 I                                        9  120FFRECL
0036 I*
0037 I* Workstation info data structure
0038 I*
```

```
IINFDS       DS
I                                     *STATUS  STATUS
I*
I* Local data area containing initial file open parameters
I*
I           UDS
I                                      201 208 NAME
I                                      209 212 PARMS
C/EJECT
C*
C* If file specified on procedure call, then skip initial prompt
C*
C           NAME      COMP *BLANKS              11
C   11                GOTO OPEN2
C*
C* Prompt for a file name to open
C*
C           OPEN1     TAG
C                     EXCPTPROMPT
C                     READ WORKSTN                1111
C   KG                GOTO EOJ
C*
C* Open the file
C*
C           OPEN2     TAG
C                     MOVE '*OPEN '  OP      6
C                     EXIT SUBRFA                     Call SUBRFA to open
C                     RLABL          OP
C                     RLABL          DTF   128
C                     RLABL          NAME
C                     RLABL          PARMS
C                     RLABL          FEEDBK
C*
C                     EXSR MSG                        Get message text
C                     MOVELOP        TESTA   1
C           TESTA     IFNE '*'                        If returncode bad
C                     GOTO OPEN1                       Then retry open
C                     END
C*
C                     MOVE *BLANKS   REC              Clear record buffer,
C                     MOVE *BLANKS   R50               screen buffer
C                     MOVE *ZEROS    SEG               and segment flags
C/EJECT
C*
C* Process data requests
C*
C           DATA      TAG
C*
C                     EXCPTEDIT
C                     SETOF                     KAKBKC
C                     SETOF                     KDKEKF
C                     SETOF                     KGKHKI
C                     SETOF                     KJKKKL
C                     SETOF                     KPKQKR
C                     SETOF                     KY
C                     READ WORKSTN                1111
C   KG                GOTO EOJ
C                     MOVE *BLANKS   OP               Clear opcode
C*
C* Process OPEN request
C*
C           STATUS    IFEQ 01125                      If HELP pressed
C                     MOVE '*CLOSE'  OP                Then close file
C                     EXIT SUBRFA
C                     RLABL          OP
C                     RLABL          DTF
C                     GOTO OPEN1                       Go perform open
C                     END
C*
C* Process command keys
C*
C                     EXSR CMD                        Go process the cmd
C*
C* Perform disk data management operation if one is pending
C*
```

```
0114 C           OP        IFNE *BLANKS                        If we have an op
0115 C                     EXIT SUBRFA                          Call SUBRFA to exec
0116 C                     RLABL          OP
0117 C                     RLABL          DTF
0118 C                     RLABL          RECORD
0119 C                     RLABL          KEY
0120 C*
0121 C                     MOVELOP        TESTA   1
0122 C           TESTA     IFEQ '*'                            If returncode OK
0123 C                     EXSR SAVEBN                            Then save binary
0124 C                     MOVE *BLANKS   R50                         clear buffer
0125 C                     MOVEAREC,RP    R50,1                       move recdata
0126 C                     EXSR SEGTAG                            and set seg tags
0127 C                     END
0128 C*
0129 C                     EXSR MSG                            Get possible msgtext
0130 C                     END
0131 C*
0132 C*
0133 C                     GOTO DATA
0134 C*
0135 C* End of program
0136 C*
0137 C           EOJ       TAG
0138 C                     SETON                     LR
0139 C/EJECT
0140 C*
0141 C* Retrieve possible message text
0142 C*
0143 C           MSG       BEGSR
0144 C*
0145 C                     MOVE *BLANK    MSGTXT 78
0146 C                     MOVELOP        TEST1   1
0147 C                     MOVELOP        TEST2   2
0148 C           TEST1     IFNE '*'                            If error retn
0149 C           TEST2     IFEQ '##'                            and its a sys err
0150 C                     MOVE OP        SYSERR  8              then build msg
0151 C                     MOVEL'SYS-'    SYSERR
0152 C                     MOVELSYSERR    MSGTXT
0153 C                     MOVE OP        SYSMIC                  Extract MIC
0154 C                     MOVE '1'       SYSLVL                  Set for USER1
0155 C                     EXIT SUBR23                            Retrieve msg text
0156 C                     RLABL          SYSMIC  4
0157 C                     RLABL          SYSTXT 69
0158 C                     RLABL          SYSLVL  1
0159 C                     RLABL          SYSRET  1
0160 C                     MOVE SYSTXT    MSGTXT                  Set message text
0161 C                     ELSE                                 Else we must lookup
0162 C                     Z-ADD1         X       20
0163 C           OP        LOKUPMSGKEY,X                   11  Lookup message
0164 C   11                MOVELMSG,X     MSGTXT                 If found, set it
0165 C  N11                MOVELOP        MSGTXT                  Else show wierdo
0166 C  N11                MOVE '?ERROR?' MSGTXT
0167 C                     END
0168 C                     END
0169 C*
0170 C                     ENDSR
0171 C/EJECT
0172 C*
0173 C* Process command/function keys
0174 C*
0175 C           CMD       BEGSR
0176 C*
0177 C           NEWRP     IFGT 0                              If new rec pos
0178 C           NEWRP     IFLE FFRECL                          is valid
0179 C                     MOVEAR50       REC,RP                  Copy from buffer
0180 C                     Z-ADDNEWRP     RP      40              Set new RP
0181 C                     MOVE *BLANKS   R50                     Clear buffer
0182 C                     MOVEAREC,RP    R50                     Copy to buffer
0183 C                     EXSR SEGTAG                            Set segment tags
0184 C                     END
0185 C                     END
0186 C*
0187 C           STATUS    IFEQ 01122                          If roll-up
0188 C                     MOVE '*GETN '  OP                    Then get next
```

```
0189 C                    Z-ADD1         RP      40
0190 C                    END
0191 C*
0192 C          STATUS    IFEQ 01123                      If roll-down
0193 C                    MOVE '*GETP '  OP                Then get prev
0194 C                    Z-ADD1         RP      40
0195 C                    END
0196 C*
0197 C   KA               EXSR NXTSEG                     Next rec segment
0198 C   KB               EXSR PRVSEG                     Prev rec segment
0199 C   KC               MOVE '*GETK '  OP               Get by key
0200 C   KC               Z-ADD1         RP
0201 C   KP               MOVE '*ADD  '  OP               Add record
0202 C   KP               Z-ADD1         RP
0203 C   KQ               MOVE '*DEL  '  OP               Delete record
0204 C   KR               MOVE '*UPD  '  OP               Update record
0205 C   KR               MOVEAR50       REC,RP            (copy from buffer)
0206 C   KR               EXSR RESTBN                      (restore bin data)
0207 C   KF               MOVE '*REL  '  OP               Release record
0208 C   KH               MOVE '*GETR '  OP               Get by RRN
0209 C   KH               Z-ADD1         RP
0210 C   KI               MOVE '*SBOF '  OP               Set BOF
0211 C   KJ               MOVE '*SEOF '  OP               Set EOF
0212 C   KK               MOVE '*GETF '  OP               Get first
0213 C   KK               Z-ADD1         RP
0214 C   KL               MOVE '*GETL '  OP               Get last
0215 C   KL               Z-ADD1         RP
0216 C   KY               MOVE '*FEOD '  OP               Fix end-of-data
0217 C*
0218 C                    ENDSR
0219 C/EJECT
0220 C*
0221 C* Subroutine to advance to next record segment
0222 C*
0223 C          NXTSEG    BEGSR
0224 C*
0225 C                    MOVEAR50       REC,RP           Copy from buffer
0226 C                    ADD  500       RP               Bump to next seg
0227 C          RP        IFGT FFRECL                     If past EOR
0228 C                    SUB  500       RP                Then undo
0229 C                    END
0230 C*
0231 C                    MOVE *BLANKS   R50              Clear buffer
0232 C                    MOVEAREC,RP    R50              Copy to buffer
0233 C                    EXSR SEGTAG                     Set segment tags
0234 C*
0235 C                    ENDSR
0236 C/SPACE 3
0237 C*
0238 C* Subroutine to backup to prev record segment
0239 C*
0240 C          PRVSEG    BEGSR
0241 C*
0242 C                    MOVEAR50       REC,RP           Copy from buffer
0243 C                    SUB  500       RP               Bump to next seg
0244 C          RP        IFLT 1                          If past BOR
0245 C                    Z-ADD1         RP                Then anchor at 1
0246 C                    END
0247 C*
0248 C                    MOVE *BLANKS   R50              Clear buffer
0249 C                    MOVEAREC,RP    R50              Copy to buffer
0250 C                    EXSR SEGTAG                     Set segment tags
0251 C*
0252 C                    ENDSR
0253 C/SPACE 3
0254 C*
0255 C* Subroutine to compute segment tags
0256 C*
0257 C          SEGTAG    BEGSR
0258 C*
0259 C                    MOVE *BLANKS   SEG              Clear seg array
0260 C                    Z-ADDRP        TAG     40       Clear seg array
0261 C          1         DO   10        X                For each seg tag
0262 C          TAG       IFLE FFRECL                      If EOR not reached
0263 C                    Z-ADDTAG       SEG,X   40        Set tag
```

```
C                     ADD  50        TAG                 Bump tag
C                     END
C                     END
C*
C                     ENDSR
C/EJECT
C*
C* Save binary data and clear in record buffer
C*    This is to prevent non-displayable characters from being output.
C*    Anything less than blank (X'40') is nondisplayable
C*
C           SAVEBN    BEGSR
C*
C           1         DO   FFRECL    C       40
C           REC,C     IFLT ' '                          If nondisplayable
C                     MOVE REC,C     BIN,C               Then save bin data
C                     MOVE ' '       REC,C               And put marker char
C                     ELSE                              Else
C                     MOVE ' '       BIN,C               Clear bin data
C                     END
C                     END
C*
C                     ENDSR
C/EJECT
C*
C* Restore binary data and clear in record buffer
C*    This is to ensure that nondisplayable data is not corrupted
C*
C           RESTBN    BEGSR
C*
C           1         DO   FFRECL    C       40
C           BIN,C     IFLT ' '                          If nondisplayable
C                     MOVE BIN,C     REC,C               Then rest bin data
C                     END
C                     END
C                     ENDSR
C/EJECT
OWORKSTN E                PROMPT
O                                   K8 'FLEDIT00'
O                         NAME       8
O                         PARMS     12
O                         MSGTXT    90
O*
O        E                EDIT
O                                   K8 'FLEDIT01'
O                         NAME       8
O                         PARMS     12
O                         KEY      111
O                         FFRECLZ  115
O                         FFRUSDZ  123
O                         SEG,1 Z  127
O                         R50,1    177
O                         SEG,2 Z  181
O                         R50,2    231
O                         SEG,3 Z  235
O                         R50,3    285
O                         SEG,4 Z  289
O                         R50,4    339
O                         SEG,5 Z  343
O                         R50,5    393
O                         SEG,6 Z  397
O                         R50,6    447
O                         SEG,7 Z  451
O                         R50,7    501
O                         SEG,8 Z  505
O                         R50,8    555
O                         SEG,9 Z  559
O                         R50,9    609
O                         SEG,10Z  613
O                         R50,10   663
O                         MSGTXT   741
** Message keys and text
#41   Permanent I/O error
#42   End or beginning of file
#43   Invalid operation code
```

```
#44   Record not found
#45   Record update attempted before input
#48   Invalid relative record number
#49   Invalid data record
#50   Update key error
#53   Duplicate relative record number
#60   Duplicate key
#61   Duplicate key in another index
#62   Key out of sequence
#63   Invalid key length
#70   File is full
#75   Undefined access type
#99   File not opened
#UB4I Record update attempted before input
#BADOPBad operation code
#DTFE DTF field is not 256 bytes long when calling SUBRFA
#NOTC File open attempted, but DTF passed to SUBRFA is not closed
#RLERRRecord length of file opened exceeds length of RPG record buffer
#UNOP File is unopened
```

Figure 8-2

Screen format member FLEDITFM

```
*         1   .    .  2          3    .     4          5          6    .     7     .    8
     SFLEDITOO            YY                                            G
     DFA0001      1  1  2Y  Y           Y     Y                CO
     DFA0002      9  1  4Y                                     CFilename
     DFL0004      8  114Y   Y
     DFL0005     16  132Y                                      COpen parameters
     DFL0006      1  149Y   Y                     Y
     DFL0007      2  151Y   Y                     Y
     DFL0008      1  154Y   Y                     Y
     DFA0001     18  156Y                                      C            FLEDIT
     DFL0009     30  241Y                                      C       /  !   \        X
     D
     DFL0012     30  341Y                                      CI=Input    !     K=Keyed X
     Daccess
     DFL0013     30  441Y                                      CO=Output  Share Level  X
     D
     DFL0015     30· 541Y                                      CU=Update  (RR, RM, MM, X
     DMR, NO)
     DFA0001     7824  2Y                 Y
     DSFLEDIT01            YN                                          235DEMNSTUVWX
     DDID          1  1  2Y  Y           Y     Y                C1
     DDFA0001      9  1  4Y                                     CFilename
     DDFL0004      8  114Y
     DDFL0005     16  132Y                                      COpen parameters
     DDFL0006      1  149Y                     Y
     DDFL0007      2  151Y                     Y
     DDFL0008      1  154Y                     Y
     DDFA0001      6  168Y                                      CFLEDIT
     DDFA0001      4  2  9Y                                     CKey
     DDFL0016     99  214Y   Y                    Y  Y
     DDFA0002      8  511Y                                      CRecLeng·
     DDFA0004      4  520Y
     DDFA0001      8  544Y                                      CRecords·
     DDFA0002      8  553Y
     DDFA0001      4  6  6   YN   Z                 Y
     DDFA0001     50  611Y                                      C1        10         20    X
     DD      30         40         50
     DDFA0003      4  7  6Y
     DDFA0002     50  711Y   Y            Y          Y  Y
     DDFA0004      4  8  6Y
     DDFA0003     50  811Y   Y            Y          Y  Y
     DDFA0005      4  9  6Y
     DDFA0004     50  911Y   Y            Y          Y  Y
     DDFA0006      410  6Y
     DDFA0005     501011Y   Y            Y          Y  Y
     DDFA0007      411  6Y
     DDFA0006     501111Y   Y            Y          Y  Y
     DDFA0008      412  6Y
     DDFA0007     501211Y   Y            Y          Y  Y
     DDFA0009      413  6Y
     DDFA0008     501311Y   Y            Y          Y  Y
     DDFA0010      414  6Y
```

```
DDFA0009    501411Y  Y            Y          Y Y
DDFA0011     415 6Y
DDFA0010    501511Y  Y            Y          Y Y
DDFA0012     416 6Y
DDFA0011    501611Y  Y            Y          Y Y
DDFA0023   56017 1Y                               C                    X
DD                                                             Help-Open a X
DDnew file  ShiftCmd3-Add record                                    Enter-X
DDUpdate buffer    ShiftCmd4-Delete record       Cmd9-Set to BOF         X
DD  Roll-Read next/prev   ShiftCmd5-Update record      Cmd10-Set to EOF   X
DD        Cmd1-Next rec segment      Cmd6-Release record      Cmd11-Get fiX
DDrst record      Cmd2-Prev rec segment      Cmd7-End program         Cmd12X
DD-Get last record       Cmd3-Get by key           Cmd8-Get by RRN        X
DD  Cmd24-Fix end of data
DDFA0001    7824 2Y
```

Figure 8-3

Procedure FLEDIT

```
* File editor
// LOCAL OFFSET-201,DATA-'?1?',BLANK-8       File name
// LOCAL OFFSET-209,DATA-'?2'U'?',BLANK-1    Access       Default  Update
// LOCAL OFFSET-210,DATA-'?3'MM'?',BLANK-2   Share level  Default  SHRMM
// LOCAL OFFSET-212,DATA-'?4?',BLANK-1       Keyed flag   Default  Unkeyed
// MEMBER USER1-##MSG1
// LOAD FLEDIT
// RUN
```

Figure 8-4a

Valid file opcodes

***ADD:**	Add a record to the file	***GETP:**	Get previous
***DEL:**	Delete a record	***GETR:**	Get by RRN (CHAIN direct)
***GETA:**	Get a record by key above	***GETF:**	Get the first record
***GETC:**	Get current record	***GETL:**	Get the last record
***GETD:**	Get next duplicate key	***UPD:**	Update the last record read
***GETE:**	Get keyed equal or high (SETLL)	***REL:**	Release the last record read
***GETK:**	Get keyed (CHAIN)	***SBOF:**	Set to beginning of file
***GETN:**	Get next	***SEOF:**	Set to end of file

The OP field may contain an error code after returning from SUBRFA. The error code always begins with '#'. The standard codes are shown in the compile-time table at the end of program FLEDIT. If the error code starts with '##', the remaining four digits are an SSP Message Identification Code (MIC) for a system message described in the IBM System Messages publication.

Figure 8-4b

RLABL code structures for closing a file

```
EXIT SUBRFA
RLABL        OP
RLABL        DTF
```

OP Contains the operation code, "*CLOSE," left justified.

DTF The DTF field associated with the file you are closing. After you close the file, you can open a different file to take its place in the same DTF field. By using different DTF fields, you can open numerous files simultaneously.

Figure 8-4c

RLABL code structures for opening a file

```
EXIT  SUBRFA
RLABL       OP          6
RLABL       DTF       128
RLABL       NAME        8
RLABL       PARMS       4
RLABL       FEEDBK
```

OP Contains one of the operation codes associated with file manipulation functions. For example, to open the file, enter "*OPEN" in the "OP" field.

DTF A 128-byte field that contains the "Define- the-file" control block for the file being opened. This field must be unique for each file that is opened within the program and cannot be an array or an array element. Never change the content of this field, because it is used by SUBRFA internally.

NAME The name of the file to be opened. This field must be the exact label of the file as it appears on the S/36 VTOC.

PARMS Contains the "open" parameters xyyz, where:

x = Type of processing
- I = Input
- U = Update
- O = Output

yy = Share level
- RR = Read/Read
- RM = Read/Modify
- NO = No Sharing
- MM = Modify/Modify
- MR = Modify/Read

z = Keyed access
- K = Keyed access

FEEDBK The name of the data structure that will receive information about the file attributes after the file is opened. The format of this data structure is

positions 1 - 8 number of records used
positions 9 - 12 record length
positions 13 - 20 file capacity in records

Like data from any file, only code and use what you need for your particular application.

Figure 8-4d

RLABL code structures for accessing a file

```
EXIT SUBRFA
RLABL        OP
RLABL        DTF
RLABL        record
RLABL        key
```

OP Contains one of the operation codes, left justified.

DTF The name of the DTF field that is associated with the file you are accessing.

record Name of the field or data structure that will contain the record being accessed.

key Contains the key data for indexed file operations or the relative record number (left justified, eight digits) for RRN operations.

Figure 8-5

FLEDIT initial display

```
Filename  EDITPROF          Open parameters: ...                FLEDIT
Key
RecLeng   512                     Records          1
     1        10        20        30        40        50

Help-Open a new file   Shift Cmd3-Add record
Enter-Update buffer    Shift Cmd4-Delete record      Cmd9-Set to BOF
Roll-Read next/prev    Shift Cmd5-Update record      Cmd10-Set to EOF
Cmd1-Next rec segment        Cmd6-Release record     Cmd11-Get first record
Cmd2-Prev rec segment        Cmd7-End program        Cmd12-Get last record
Cmd3-Get by key              Cmd8-Get by RRN         Cmd24-Fix end of data
```

Re-creating Subroutine SUBRFA

If you don't have assembler subroutine SUBRFA, you can re-create it with procedure MKSUBRFA (you don't need IBM's Assembler Language Program Product to install SUBRFA). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRFA. You need to run MKSUBRUF only once because SUBRFA is subsequently linked into program FLEDIT when it is compiled.

```
// * Re-creating R-module SUBRFA in library #RPGLIB
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201 DATA-'00000023'     Number of #MAINT records
```

Continued

```
// LOCAL OFFSET-209,DATA-*
'D9E2E4C2D9C6C1404000000E000000000000060400000000990007200000009787'
// LOCAL OFFSET-273,DATA-*
'0926212831000000803B00000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRFA member to insert object code
// LOAD $FEFIX
// RUN
HDR  3858 SUBRF00000
PTF  2B16 RSUBRFA,99,,#RPGLIB
DATA 0EE3 00 0000 E208E2E4C2D9C6C1003EF2052B00000000000000000000000000000000000000
DATA 96A8 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 37B8 00 0040 E3323F24340840573401404C34024050F4000AF4000F8A25007A1030C2A10000
DATA A5CA 00 0060 350240572C0143B7052C0043B3030F0143B74383B50202002F2D28231F080703
DATA 8AD7 00 0080 E32E3F53340243BF2C0543BD000E01405743E80D0543BD440CC08140690D0543
DATA DB0B 00 00A0 BD4412C08141DE350240572C0143D0028D0202002A26221E1C1814120E0C0703
DATA 35EE 00 00C0 E32E3F8243F9F201130E01405743E62C0143D2042C0143D006F2870F2C0043D2
DATA E2E2 00 00E0 003C0043D10F0143D043D20E01430243E20E0100002C2A2624201B130E0A0801
DATA C1F6 00 0100 E3313F84405743E43C0043CCC202422F2C0043CC068D050543BDF28109E20207
DATA A46F 00 0120 BD0000F10116BD0000F201100C0543D842C00E0140570031 2D28150F08070301
DATA 6C78 00 0140 E32F3FE443E4C08740583501405 70E01405743E41C0043B500350243B79C014F
DATA AD60 00 0160 02BF014F43B58F014F43B78F014F44060D0543B0002F28262118130F0D090501
DATA 07AE 00 0180 E3314016428FF201181C004002003D074002F204043C074002760102AF073535
DATA AA42 00 01A0 9C0735000C0543D842D28D011443D2C0844058BC010C00002E2A2523140D0801
DATA 0BCC 00 01C0 E331404843D08C011F43D08C000E43CCF4010C00BD400DF2811CC2A100003501
DATA DC80 00 01E0 438F0C0543D842DE280243D40028034360 0C087405800312C2723211D080601
DATA 8CAE 00 0200 E332407BC2A10000C2A20000F68001C0870000C2A1000035014 38F4C050043D8
DATA 416E 00 0220 C0874049 3D7F43B3F202100C0543D842C60E01405743EE0032302C2A231F1B16
DATA F2E5 00 0240 E32E40AAC0874058350243878801 0FF290100C0543D842CC0E01405743EEC087
DATA C953 00 0260 4068AF7F7F7F8C616143B1350140570E014057 0000002E2A261D181713110703
DATA 67B3 00 0280 E33240DD43E47601029C071C008C07354404350140570E01405743E47601021C
DATA 12F5 00 02A0 0343CE008CC00F3DE443C8F201038A2C0F3D0643CBF2010030261E1716110D01
DATA 13D9 00 02C0 E32F410D038A200F3DD243CEF20103BA0210340242E90D0143CD4414F201043C
DATA FDAF 00 02E0 0043110D0143CD4416F201043C0143110D0143CD0000002F2B24221E17151107
DATA C164 00 0300 E330413E4418F201043C0943110D0143CD441AF201043C0243110D0143CD441C
DATA 5109 00 0320 F201043C034311C2A242E5F401040E3D0042F6F2B1002E26221B19150E0C0801
DATA 8EB0 00 0340 E32C416B280C0643D842E4080243 0642F5080343D842F5080243D742F6080343
DATA 4F30 00 0360 D842F60E01405743EAC0874058350242E9002C2824221E1C181612100C0A0604
DATA 4AA4 00 0380 E331419D85A24D85A21FC2A10000350140570E01405743EA7601062C02422E38
DATA 1A63 00 03A0 C08741FE4C070743E02C01422E3E3C00422CC08741FE00312D28241F1A13110D
DATA 3633 00 03C0 E33441D24C030B43E02C02422E4FC08741FE4C071343E0C2A20000350242E9B5
DATA 189D 00 03E0 A14D75A11F9C01143E3DD243CEF201089C0051429C015344C200281A120D0804
DATA 6E68 00 0400 E33042 03A10000F401040 2C0874049350243B7F401040334A242E93A4042ECC2
DATA CCE0 00 0420 A242E5F401040D3B4042ECC087404934 08422A07070000002E2A261E1A160E0A
DATA 52BA 00 0440 E327422843E043E03C18422B060743E043EC0E02422E427EF220043A0143E03F
DATA AB26 00 0460 01422BC001420CC08700000000000000000000000000221E1A13110D08070301
DATA AA08 00 0480 E33A42895CC1C4C44040405CC4C5D34040205CC4C5D3D940215CC8C5D6C4400C
DATA 611D 00 04A0 5CC7C5E3C1408D5CC7C5E3C340855CC7C5E3C5408C5CC7C5E3C440865CC7C500
DATA E811 00 04C0 E33A42A4E3C840B15CC7C5E3D240805CC7C5E3D340825CC7C5E3D540835CC7C5
DATA 9E21 00 04E0 E3D740845CC7C5E3D940895CD9C5034040015CE2C2D8C640055CE2C5D8C64000
DATA 0574 00 0500 E33A42DF045CE4D7C44040485CE4D7C4D94049007BE4C2F4C9407BC2C1C4D6D7
DATA 6B67 00 0520 7BC4E3C8C54078D5D6E3C34078D903C5D9D97BE4D5D6D7407BF0F04040407B00
DATA E848 00 0540 E33A431A78F0F0F0F0C1E2B000000000004000000000000000000000000000000
DATA DD64 00 0560 0000000000000000000000000000000000000000009000000000404040404000
DATA 9AB6 00 0580 E33A4366404040404040404040404040000000000000000000000000000000000
DATA 7B90 00 05A0 00000000000000000000000000000000000000000000000000A0000000800000
DATA 544C 00 05C0 E33A43900080FFFFB0000040009C000010000AC4C1E3C1C1D9C5C1B000000001
DATA 9466 00 05E0 80FFFF00000000000080FFFF0000000000000000FFFF00000000000000000000
DATA 0886 00 0600 E324438600000000000000000000000000000000000000000000000000000000
DATA BF62 00 0620 0000000000000000000000000000000000000000000000000000000000000000
DATA 1F15 00 0640 E33A441B000100030004000600070009000D6FC8D9D3E5C4E2C9FFFFFF000000
DATA 0AEB 00 0660 F0F0F0F0F0F0F0F0F100625CD6D7C5D5405CC3D3D6E2C6D9D4D9D9D4D4D4D9D500
DATA 4899 00 0680 E300441CD6000000000000000000000000000000000000000000000000000000
```

Continued

```
DATA 4876 00 06A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 2CFC 00 06C0 C5FFFF1C00000000000000000000000000000000000000000000000000000000
DATA 40DB 00 06E0 0000000000000000000000000000000000000000000000000000000000000000
END  FE33
```

Retrieving a File's Users

by Perry Gardai

program by Matthew Henry

Code on diskette:
Procedure TESTUF
RPG program TESTUF
Assembler subroutine SUBRUF

When your S/36 file maintenance chores are stymied by a "file in use" message, use this utility to identify who is using the file. The S/36 utility TESTUF determines which workstations or jobs are using a particular file and gives you a tool upon which to build new utilities.

Most S/36 programmers know how frustrating it is to try to perform file maintenance functions such as a DELETE or variations of COPYDATA, RENAME, and SAVE that require a dedicated file. Invariably, an unknown culprit is using the file you need. Unfortunately, IBM provides no way to determine who is using a particular file. To solve this problem, we present the S/36 TESTUF utility, which offers an easy and effective method of determining file use from any terminal on the system. The TESTUF utility allows you to determine the users of a file, including each active procedure and program and the job start time, as well as the file sharing level in effect for each user. The procedure does not require the target file to be dedicated.

Procedure TESTUF (Figure 8-6) serves as the user interface by calling program TESTUF (Figure 8-7), which calls subroutine SUBRUF. For ease of access, procedure TESTUF and program TESTUF should be stored in #LIBRARY.

To use the TESTUF utility, simply key in

```
TESTUF filename
```

where *filename* (parameter 1) is the name of the file to be checked for current users. Procedure TESTUF loads the file name into the LDA, beginning in position 247. The TESTUF utility uses LDA positions 201 through 262 to avoid conflict with the LDA positions IBM's POP uses. Procedure TESTUF initializes parameter 2 to zero and loads it into the LDA starting in position 255, where it serves as a loop counter. Procedure TESTUF then calls program TESTUF, a one-cycle RPG program that calls SUBRUF via the EXIT operation and three RLABL statements. The first RLABL statement contains the file name you specified. Subroutine SUBRUF retrieves information about one user of this file and stores the user information in data structure JOBDS, named in the third RLABL statement. (This data structure must be at least 47 bytes long to hold all the information SUBRUF returns. If the data structure is not long enough, SUBRUF will not return any data.)

Because the specified file could have several users, SUBRUF allows repetitive calls to retrieve information about each of them. The second RLABL statement, JOB#, specifies the user for which SUBRUF should return information. Field JOB# contains 0 to return information about the first job using the specified file, 1 for the second job, 2 for the third job, and so on. After calling SUBRUF, program TESTUF copies the contents of the JOBDS data structure (information about a user of the file) into LDA positions 201 through 262 via the LJINFO field, and procedure TESTUF displays this user information on your workstation screen. Then procedure TESTUF increments the counter, parameter 2, and repeats the process until position 209 of the LDA (corresponding to field JOBNAM in data structure JOBDS) is blank. This loop is repeated as often as jobs are found running from the specified file and results in a scrolling screen that displays messages containing information about all users of the specified file.

The JOBDS data structure returned by SUBRUF contains information about the file sharing level (access privileges) for each user. The field SHRLVL is a one-digit code with the following meaning:

Code	**Sharing Level**
0	Read/Modify
1	Read/Read
2	Modify/Read
3	No Sharing
4	Modify/Modify

Program TESTUF uses an array to translate this numeric code into the standard alphabetic notation used by the SSP to designate file sharing levels (e.g., SHRMM is the notation that designates a sharing level of Modify/Modify).

When position 209 of the LDA is blank (i.e., no other jobs are using the specified file), the procedure performs a final test of parameter 2. If parameter 2 is 0 at this time, no workstation or job is using the specified file, and a message is issued accordingly. (If parameter 2 is a value other than 0, no message is issued in addition to the file user information.) In either case, procedure TESTUF then terminates.

As with any user members stored in an IBM-supplied library (e.g., #RPGLIB or #LIBRARY), you should remember that subroutine SUBRUF, program TESTUF, and procedure TESTUF will be removed from the system each time you install a new release of SSP. Therefore, you should keep a copy of all the components of this utility in your tool kit library so you can readily replace them after you install a new release.

The TESTUF utility demonstrates tool building — it uses a core tool (SUBRUF) to create a new tool. You can implement a core tool as a subroutine to incorporate into other tools, to build completely new tools, or to use one tool in different ways. For instance, you could incorporate the TESTUF utility directly into the IBM-supplied DELETE, COPYDATA, or SAVE procedures to show a list of jobs using a file before you get the "file in use" message.

The TESTUF utility can help you in your file maintenance chores by identifying who is using the file that you need to access. And you also can make your programming efforts more effective if you use these tool-building concepts.

Figure 8-6

Procedure TESTUF

```
* Find out who's using a file
// INFOMSG YES
// LOCAL OFFSET-247,DATA-'?1R?',BLANK-8
// EVALUATE P2,3-0
// * 'The following jobs are using file ?1?,
// TAG LOOP
// LOCAL OFFSET-255,DATA-'?2?'
// LOAD TESTUF
// RUN
// IF ?L'209,1'?-  GOTO DONE
// * 'Job ?L'209,8'?, User ?L'201,8'?, Proc ?L'217,8'?, +
      Running ?L'233,8'?(?L'225,8'?), DISP-?L'258,5'?'
// EVALUATE P2,3-?2?+1
// GOTO LOOP
// TAG DONE
// IF ?2?-000 * '(nobody)'
// PAUSE
```

Figure 8-7

Program TESTUF

```
*.        1         2         3         4 . .     5       . 6     .. 7 .      8
H      084                          8       1                             TESTUF
 *
 *- This program tests SUBRUF by retrieving job information for a job
 *    using a specified file
 *
E                       SHR     10  10  5
I             UOS
I                                           201 248 LJINFO
I                                           247 254 FILNAM
I                                           255 2570JOB#
I                                           258 262 SHRTXT
IJOBDS          OS
I                                             1   8 USERID
I                                             9  16 JOBNAM
I                                            17  24 FSTPRC
I                                            25  32 CURPRC
I                                            33  40 PRGNAM
I                                            41  460JSTIME
I                                             1  46 JINFO
I                                            47  470SHRLVL
C                       SETON                         LR
C                       EXIT SUBRUF
C                       RLABL          FILNAM
C                       RLABL          JOB#
C                       RLABL          JOBDS
C                       MOVE JINFO     LJINFO
C             SHRLVL    ADD  1         X        20
C                       MOVE SHR,X     SHRTXT  5
** Share levels
SHRRMSHRRRSHRMRNOSHR                      SHRMM
```

Re-creating Subroutine SUBRUF

If you don't have assembler subroutine SUBRUF, you can re-create it with procedure MKSUBRUF (you don't need IBM's Assembler Language Program Product to install SUBRUR). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRUF. You need to run MKSUBRUF only once because SUBRUF is subsequently linked into program TESTUF when it is compiled.

Continued

```
// * 'Re-creating R-module SUBRUF in library #RPGLIB
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'     Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9E4C640400000004000000000000060400000000990002200000000087'
// LOCAL OFFSET-273,DATA-+
'1001221031000000803B000000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRUF member to insert object code
// LOAD $FEFIX
// RUN
HDR  38D0 SUBRU00000
PTF  1CDD RSUBRUF,99,,#RPGLIB
DATA 144B 00 0000 E20BE2E4C2D9E4C60000000114000000000000000000000000000000000000
DATA 9C0E 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 9527 00 0040 E3300030340800F2340100EA340200EEF68080350100F27502022C07010A0075
DATA 5617 00 0060 02052C020110004D010A0102C082000E75020C34A20000002B27211912080703
DATA 44EE 00 0080 E33200630113BC4025AC242425F4000F8A700036A100F5F2813375A2BA34A101
DATA 2A1F 00 00A0 0D36A200F5F2811B85A11F4D070B010AF2010A07020110000000322B201C1201
DATA 7610 00 00C0 E334009B00FEC082007EB5A21CF1872235A1010D75A111F1873AC087000EE2A1
DATA 7A4F 00 00E0 0035A201139C002E108AF02E35A1010D9C07070C9C0717749C002B20190F0501
DATA F873 00 0100 E33800D1071F7C9C0727849C0109698C050F00FB98020A6A98030B6A98020C6B
DATA C4BB 00 0120 98030D6B98020E6C98030F6C8C052D00FB98022B6598032965980022A66002C0F
DATA 41CB 00 0140 E33001029803286698022C679B0320670E0100F20100F68000C2A10000C2A200
DATA 3FE1 00 0160 00C0870000000000F0F0F0F0F0F0F0F01000D002E0000000000000000000110F
DATA AB1E 00 0180 C5FFFF0200000000000000000000000000000000000000000000000000000000
DATA AAE7 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 8539 00 01C0 615C000000000000000000000000000000000000000000000000000000000000
DATA A400 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
END  5657
```

Displaying Record Locks

by Gary T. Kratzer

program by Mel Beckman

Code on diskette:
Procedure SHOWUR
RPG program SHOWUR
Assembler subroutine SUBRUR
Screen format member SHOWURFM

Use utility SHOWUR to determine which record is locked and which terminal is responsible.

The record lock is a fact of life in S/36 shops. Because the S/36 was designed for multiple users in an interactive environment, operators constantly update records in master files. Quite often, different operators try to update the same record at the same time. The system looks unfavorably on such attempts, and it responds by "freezing" any terminal that tries to access a record already in use.

In the interactive environment, a record's integrity depends on up-to-the-minute information. When Operator A updates an address in a record, the updated record writes over any previous version of the record in the master file. Operator B then uses the updated version when later changing the phone number in the same record. This update likewise replaces the version previously supplied by Operator A, and your master file record now contains both the correct address and the correct phone number. The interactive environment, by design, cannot accommodate simultaneous record updating.

To minimize the chances of record locks, interactive programmers take a number of tacks. A S/36 program might include a command that releases a record immediately after it has been read. Or the program might make use of "busy flags" to warn operators that the record they want to access is already in use. A section of IBM's *System/36 Concepts and Programmers Guide* (SC21-9019) is devoted to avoiding record locks. But despite such "tricks," record locks are common at S/36 sites.

Workstations can freeze up for many reasons — a record lock being but one. When a workstation freezes, your first task is to determine the cause. If all workstations are inhibited, and you can't invoke system console mode, you probably do not have a record lock. But if only certain workstations are frozen, and those workstations share some or all of the same files, a record lock is likely.

So what do you do when you discover a potential record lock? How do you determine which record is locked and which job is responsible? Most S/36 sites don't even try to answer these questions. Instead, they commonly "cure" the record lock by asking all users (including the operator using the record that others have tried to access) to end their jobs. The coveted record is released along with all other records, and any frozen terminals become functional again.

This approach works but at times is problematic. If a record lock occurs in the midst of a giant system update that takes several hours, you don't want to forsake the update to get one or two frozen terminals up and running. Having all users end their jobs also is not convenient when workstations are spread out over several floors or several buildings. So isn't there a better way?

Have no fear! Utility SHOWUR is here! Utility SHOWUR displays information about records that a particular job uses and, as a result, helps you determine the source of a record lock. All you need to do is determine which file(s) the operator of the frozen terminal is trying to use, and SHOWUR does the rest. Utility SHOWUR comprises program SHOWUR (Figure 8-8), screen format member SHOWURFM (Figure 8-9), procedure SHOWUR (Figure 8-10), and assembler subroutine SUBRUR.

To use the utility, simply key in the letters SHOWUR, followed by the name of the file you're interested in. The resulting screen (Figure 8-11) displays a list of jobs using that particular file, as well as certain related information. The Roll keys let you page through the entries. If no data is shown on the screen, the specified file either is not on the system or is not being used by any tasks. If the file you try turns out not to be the culprit, you may change the file name to display additional files.

Three columns in the display indicate where a record lock may exist. Column RRN shows which of the file's records the job has last read. If the job has not released the record after reading it with an intent to update, a Y will appear in column *Owned.* If other jobs are trying to use the same record, a Y appears in column *Waiting.* In such instances, as illustrated by the matching RRNs in Figure 8-11, you have a record lock. The other jobs then must wait until the job owning the record has released it before they can acquire it.

When would such a situation occur? A typical scenario involves an operator who brings up a customer's record to change the address, but who goes to lunch without releasing the record. If another operator tries to bring up the same customer's record at this time, his or her terminal becomes frozen — because the record is "locked" on the first operator's screen. With the information provided by utility SHOWUR, you easily can rectify the situation. The first operator (or an authorized substitute) need only complete the update and release the record, thereby "thawing" the second operator's terminal. Be aware, however, that another job may already be waiting for the record in question — in which case you would again have a record lock.

SHOWUR can be a useful weapon in your computing arsenal. With this utility, you can conquer one problem typical of a multiuser environment. So next time your system freezes up, give SHOWUR a try. You'll save yourself hours on the phone and miles of legwork, and you'll have your users up and running again in no time.

Figure 8-8

Program SHOWUR

```
*... ... 1 ... . 2     ... 3 .   . 4 .. . . 5      .. 6  . .  7 .  . . 8
H    064                      B       1                              SHOWUR
F*
F* By. Mel Beckman, 10/01/87
F*
F* Display a list of all records for use in a specified file
F*
FWORKSTN CD  F    2000            WORKSTN
F                                             KINFDS EXCPDS
E                   LIN        20 80                Screen lines
I*
I* Screen input
I*
IWORKSTN NS
I                                       1   8 FILNAM
I*
I* Data structure returned by SUBRUR
I*
IRECDS      DS
I                                       1   8 USERID
I                                       9  16 JOBNAM
I                                      17  24 FSTPRC
I                                      25  32 CURPRC
I                                      33  40 PRGNAM
I                                      41  480RRN
I                                      49  49 FLAGS
I*
I* Screen line data structure
I*
I           DS
I                                       1  80 SCREEN
I                                       1   8 SJOBNA
I                                      11  18 SUSERI
I                                      21  28 SFSTPR
I                                      31  38 SCURPR
I                                      41  48 SPRGNA
I                                      51  580SRRN
I                                      63  63 SOWNED
I                                      71  71 SWAITG
I*
I* Workstation status data structure
```

```
I*
IEXCPDS      DS
I                                       *STATUS  STATUS
I*
I* LDA contains name of initial file
I*
I            UDS
I                                        201 208 FILNAM
C/EJECT
C*
C* Main event loop
C*
C           EOJ       DOUEQ'Y'                          Do until EOJ
C*
C                     EXSR PAGE                          Build a screen page
C                     EXCPTSCRN01                        Display it
C                     READ WORKSTN                  1111 Read the screen
C   KG                MOVE 'Y'       EOJ     1           If Cmd7, set EOJ
C*
C           FILNAM    IFNE OLDNAM                        If name changed
C                     MOVE FILNAM    OLDNAM  8            Save old name
C                     Z-ADD0         SEQ#    30           Reset SEQ#
C                     END                                End IF
C*
C           STATUS    IFEQ 01122                         If roll-up
C           EOF       IFNE 'Y'                            If not EOF
C                     Z-ADD16        SEQ#                  then bump SEQ#
C                     END                                 End IF
C                     END                                End IF
C*
C           STATUS    IFEQ 01123                         If roll-down
C                     SUB  16        SEQ#         11      Then unbump X
C   11                Z-ADD0         SEQ#                  Adjust underflow
C                     END                                End IF
C*
C                     END                               End DO
C*
C* End of program
C*
C                     SETON                     LR
C/EJECT
C*
C* Page routine
C*   Build a page of data for output in the LIN array
C*
C           PAGE      BEGSR
C*
C                     Z-ADDSEQ#      X       30         Set starting point
C                     MOVE *BLANKS   LIN                Clear line array
C                     MOVE *BLANKS   MSGLIN 70          Clear message line
C                     MOVE *BLANK    EOF     1          Clear EOF flag
C*
C                     DO   20        Y       30         Do 20 times
C                     EXIT SUBRUR                        Get record user
C                     RLABL          FILNAM               (name of file)
C                     RLABL          X                    (sequence #)
C                     RLABL          RECDS                (data structure)
C           JOBNAM    IFGT *BLANKS                       If valid name
C                     EXSR LINE                           Build a line
C                     MOVEASCREEN    LIN,Y                Store it
C                     ADD  1         X                    Bump seq#
C                     ELSE                               Else
C                     MOVE 'Y'       EOF                  Set EOF flag
C                     MOVEL'**End**' MSGLIN               Show EOF msg
C                     END                                End IF
C                     END                               End DO
C*
C                     ENDSR
C/EJECT
C*
C* Line routine
C*   Build a screen line
C*
C           LINE      BEGSR
C*
C           USERID    IFEQ *BLANKS                       If no user-ID
```

```
C                       MOVEL'MRT JOB' SUSERI          Then it's a MRT
C                       ELSE                          Else
C                       MOVE USERID    SUSERI          It's a user
C                       END                           End IF
C*
C                       MOVE JOBNAM    SJOBNA         Copy jobname
C                       MOVE FSTPRC    SFSTPR              first proc
C                       MOVE CURPRC    SCURPR              current proc
C                       MOVE PRGNAM    SPRGNA              prog name
C                       MOVE RRN       SRRN                RRN
C                       TESTB'6'       FLAGS       11 If owned bit on
C   11                  MOVE 'Y'       SOWNED          set "owned"
C  N11                  MOVE ' '       SOWNED          else clear it
C                       TESTB'7'       FLAGS       11 If waiting bit on
C   11                  MOVE 'Y'       SWAITG          set "waiting"
C  N11                  MOVE ' '       SWAITG          else clear it
C*
C                       ENDSR
OWORKSTN E                  SCRN01
O                                     K8 'SHOWUR01'
O                           FILNAM     8
O                           LIN     1608
O                           MSGLIN  1678
```

Figure 8-9
SHOWURFM screen format member

```
* . . 1 . . . . 2 . . . . 3 . . . . 4 . . . . 5 . . . . 6 . . . . 7 . . . . 8
     SSHOWUR01           NY                                AG15
     DFA0010    23 119Y                            CRecords in use for file
     DFA0001     8 143Y  Y                   Y
     DFA0001     8 2 1Y                    Y        CJob name
     DFA0002     8 211Y                    Y        C  User
     DFA0003     8 221Y                    Y        C1st Proc
     DFA0004     8 231Y                    Y        CCur Proc
     DFA0006     8 241Y                    Y        C  Prog
     DFA0007     8 251Y                    Y        C   RRN
     DFA0008     5 261Y                    Y        COwned
     DFA0009     7 268Y                    Y        CWaiting
     DFA0003  1600 3 1Y
     DFL0023    7023 2Y          Y
     DFL0024    7924 2Y                             CRoll keys-page         X
     D          Enter-update              Cmd7-End program
```

Figure 8-10
SHOWUR01 sample screen

```
                  Records in use for file CUSMAST
Job name    User     1st Proc  Cur Proc   Prog      RRN    Owned  Waiting
W2113118   MEL       LIBR#     FLEDIT    FLEDIT   00003241    Y
W3102215   GARY      CUSLIB    CMAINT    CMAINT   00003241            Y
W4082216   DON       CUSLIB    CMAINT    CMAINT   00000200    Y
W4082222   DON       CUSLIB    CUPDAT    CUPDAT   00001565            Y
W7120101   TRISH     CUSLIB    CDELET    CDEL01   00001565    Y








**End**
Roll keys-page                  Enter-update                  Cmd7-End program
```

Figure 8-11

Procedure SHOWUR

```
// LOCAL OFFSET-201,DATA-'111',BLANK-8
// LOAD SHOWUR
// RUN
```

Re-creating Subroutine SUBRUR

If you don't have assembler subroutine SUBRUR, you can re-create it with procedure MKSUBRUR (you don't need IBM's Assembler Language Program Product to install SUBRUR). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRUR. You need to run MKSUBRUR only once because SUBRUR is subsequently linked into program SHOWUR when it is compiled.

```
// * 'Re-creating R module SUBRUR in library #RPGLIB'
* Builds an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201 DATA-'00000103'     Number of $MAINT records
// LOCAL OFFSET-209 DATA-+
'D9E2E4C2D9E4D94094040000004000000000C000060400000000098000320000000000587'
// LOCAL OFFSET-273 DATA-+
'0930003731000000803866000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRUR member to insert object code
// LOAD $FEFIX
// RUN
HDR  3BDC SUBR000001
PTF  1CDF RSUBRUR 99, #RPGLIB
DATA 9B8C 00 0000 E2D8E2E4C2D9E4D9C2B2801 3FC00000000000000000000000000000000
DATA 748C 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 0E18 00 0040 E33226563408270E3401270634022 70AF400AF6808038012 70E7802022C0727
DATA CA86 00 0060 5E007502062C0227 1F004001042753C08226FA78020C34002E2A241C1608070 3
DATA 705F 00 0080 E331266AA227848C403 0AC7F2F3C035A1 2 7567 8A10036A1 274 6F2B12C4007 0837
DATA 8B24 00 00A0 5EF2011E75A29F36A2274 9F2811007022761 274FC08200000 2F7D281C140003
DATA AE86 00 00C0 E3312 8BC284185A204F18717CD8728FA75A17FF18733C08726FA35A127848C00
DATA 77F1 00 00E0 300F8002C0274C12811 32C022 73E000C08727 0F400 27F000000 2F78221 8150801
DATA 028A 00 0100 E33B20F52746 48702F 274F86A20785A217 8C0707 0C6C0717746C071 57 0C8C0727
DATA 8151 00 0120 346C0105 8967050F0F8 8020A6A58C3086A68020C68880300 6888020E8C000801
DATA B2B5 00 0140 E3302728888U30 FE 00EC1270E27E1F680 00C2A1000 0C2420000C0870000 34 0827
DATA 2DF3 00 0160 34070737 482 7463C1E27380807 274827 46 0E0 22 73E0000 3C2C2A262220 1C0807
DATA 588C 00 0180 E314273 8373EF22004 3A01 27463FC12773 8F 1011AC0870000 0000000000000000
DATA 6787 00 01A0 0000000000000000000000000000000000000000000000000000000000000801
DATA E958 00 01C0 E30F27680000 00FFFFFFFF0F0F1000000300009 440000000000000000000000000
DATA 0A12 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
DATA D481 00 0200 C6FFFF6800000000000000000000000000000000000000000000000000000000
DATA 863C 00 0220 0000000000000000000000000000000000000000000000000000000000000000
DATA CC86 00 0240 616C000000000000000000000000000000000000000000000000000000000000
DATA 7752 00 0260 0000000000000000000000000000000000000000000000000000000000000000
DATA 4E12 00 0280 4040404040404040404040404040404040404040404040404040404040404040
DATA 8CEA 00 02A0 4040404040404040404040404040404040404040404040404040404040404040
DATA 9CB8 00 02C0 4040404040404040404040404040404040404040404040404040404040404040
DATA 0A88 00 02E0 4040404040404040404040404040404040404040404040404040404040404040
END  8CDF
```

Finding the Last Record Number in a File

by Richard E. Green

Code on diskette:
RPG code FINDLAST

I have often needed to add records to an indexed file whose key field was a one-up number. Normally, the last sequential number was maintained in a control file record. If an additional record was to be added, an add program read the last record number from the control file record, incrementing the last record number by one to determine the next record number and updating the control file record. If the add program had an abnormal end of job, and the control record was not updated, another program had to be run to rebuild the key and to update the control record.

Partial program FINDLAST (Figure 8-12) provides a solution to this problem by using a binary search (the old "I can find any number between two numbers in ten tries" routine) to find the last record number. Program FINDLAST eliminates the need both for the control record and the entire rebuild program. Program FINDLAST divides the range of values for the last record number in half. A CHAIN operation determines in which half the last record number occurs. The split-and-check process is continued until the next record to be read equals the last record read.

This program can work with either indexed files or direct files. The only restriction on direct files is that the initial "high" number cannot be greater than the file length and that the file not be full. If the file is full, the program will incorrectly return the high value as the available record.

Figure 8-12

Partial program FINDLAST

```
*... .   1         2        . 3 .  .  4 .   . . 5    . . 6     . . 7 .   . 8
     C*                                             HILOEQ
     C                     Z-ADD99999    HIGH    50        THIS ROUTINE FINDS THE
     C                     Z-ADD0        LOW     50        NEXT AVAILABLE RECORD
     C           OVER1A    TAG
     C           HIGH      SUB  LOW      DIFF    50        HIGH CONTAINS THE
     C           DIFF      DIV  2        DIFF      H       LAST UNFOUND NUMBER.
     C           DIFF      ADD  LOW      DIFF              LOW CONTAINS THE
     C           HIGH      COMP DIF                     41 LAST FOUND RECORD
     C      41             GOTO PASS1A
     C*                                             HILOEQ
     C           DIFF      CHAININUNTFEA            40
     C      40             Z-ADDDIFF     HIGH
     C     N40             Z-ADDDIFF     LOW
     C                     GOTO OVER1A
     C*
     C           PASS1A    TAG
     C                     Z-ADDHIGH     RCDNBR  50
```

Counting Records with Same Partial Keys in Indexed files

answered by Mike Patton and Ken Sims

Q I have a keysorted indexed file on a S/36 that is approaching one million records. The key length is 14 characters long, starts in position one of the record, and takes values from 01000000000000 to 20999999999999. Duplicate keys are not allowed. Is there a quick way, without reading the entire file, to determine how many keys start with 01, how many with 02, and so on?

A You can come up with the desired tallies without reading the file if your records are sequential and evenly spaced, and if none of these sequential records has been deleted. As long as the restriction against duplicate keys is enforced via an evenly spaced series of numbers (1,2,3,4,... *n*), then a simple program fragment counts occurrences in each major group:

```
          MOVE *BLANKS   LIMIT  14
          MOVE '02'      LIMIT
LIMIT     SETLLHUGEFILE
          READPHUGEFILE    99  (EQ)
```

When the first two bytes are ignored, the record key that is read at this point contains the highest key in the 01 group. (Note that, for this solution to work, at least one record must exist in the 01 group; if no 01 record exists, the error indicator 99 indicates that the beginning of the file has been reached.)

Unless these criteria are met, there is no way to calculate the record count without reading the entire file. But you can arrive at this calculation fairly quickly by reading the file as a sequential file, ignoring the index. As you read, keep a count of the number of each record type with this program fragment:

```
FIELD1    IFEQ 01
          ADD   1          RECS01
          ELSE
FIELD1    IFEQ 02
          ADD   1          RECS02
          .
          .
          .
FIELD1    IFEQ 20
          ADD   1          RECS20
          END
          .
          .
          .
          END
```

At the end of the job, you can print/display the totals.

Reducing Sort Work File Size

by Alex Barish

When writing OCL statements for a sort job, most S/36 programmers don't bother including a // FILE statement for the sort work file. When a // FILE statement is not specified for the work file, the system allocates a work file large enough to contain all the records from the input file. This automatic file allocation can add up to a lot of wasted disk space, especially if only a fraction of the records are selected for sorting. To conserve disk space, you can use the file size substitution expression in an EVALUATE statement to calculate the needed work file size. This trick can prove invaluable if you are sorting a very large file and disk space is tight.

For example, if you want to sort file MASTER, and you know the application well enough that you're sure no more than one-third of the input records will be selected for sorting, the statement

```
// EVALUATE P63=?F'A,MASTER'?/3
```

will place the value for the required number of records in parameter 63. You then can use a statement such as

```
// FILE NAME-WORK,RETAIN-J,RECORDS-?63?
```

to allocate an appropriately sized sort work file. Just be sure that you allocate enough space. SSP ignores an EXTEND parameter on a // FILE statement for a work file.

Allocating Sort Output Files

by Robert E. Puhalla

The size of our report files varies widely over the course of a month (e.g., from zero records to several thousand records). We sort these files in our daily report jobs that run at night, but the variance in size makes it difficult to automate the sorting procedure. For example, if I use a substitution expression to allocate the sort output file, the job halts with an error if the substitution expression contains zero (i.e., no records in the file to be sorted). To get around that problem, I could specify some standard size for the output file, but specifying a large sort output file wastes disk space if the report file happens to be small, while specifying a small sort file results in too many extents (and thus slow processing) if the report file is large.

To solve this problem, I include the following two OCL statements in my job that sorts file XYZ

```
// FILE NAME-INPUT,LABEL-XYZ,DISP-SHR
// FILE NAME-OUTPUT,LABEL-ABC,RECORDS-1,EXTEND-?F'A,XYZ'?
```

Output file ABC will have at least one record but only one extent. Sometimes there may be only one (blank) record in the output file, a situation that procedure SORT will interpret as "no records to be sorted." Therefore, I place an N in position 36 of the sort's H-spec to specify that no message is to be issued when the sort procedure finds no records to sort.

Performance Differences Between SORTA and SORTR

answered by Bob Tipton

Q A "Great Sort Debate" is raging in our shop. One of my cohorts contends that the use of ADDROUT (Address Output) sort files increases the performance of sorts. I contend the ADDROUT file is a disk saving technique, not a performance improvement. Who is right?

A Your cohort is right, if you think solely in terms of the sort. An addrout sort (SORTA) can be significantly faster than a tagalong sort (SORTR). However, sorts are seldom done alone. That is, you usually sort a file to come up with a report. If you consider the aggregate time of the sort and its print program, you are right; addrout sorts conserve disk space because they store three-byte relative record addresses instead of entire records, but they degrade the performance of the job.

To illustrate, let's suppose you use an addrout sort on a file and then print a report. When the sort is finished, you have two files: an addrout file that contains the relative record addresses and the original input file. To print the report in sorted order, the print program must use the relative record addresses stored in the addrout file to chain to the input file. One — and only one — record from the input file is retrieved from disk at a time.

Thus, for every record, the system reads the addrout file to locate a record and then chains to the input file to retrieve the record. The time your cohort claims you gained by using an addrout sort is lost in this latter part of the job.

On the other hand, if you had used a tagalong sort instead of an addrout sort, the system might have taken longer to sort the records, but you would end up with a single file of actual data records for the print program to read. There would be no need to read one file and chain to another. In fact, if you had used a tagalong sort, you then could "block" the number of records the S/36 read from the input file in one disk access and thus reduce disk accesses and improve performance.

Because addrout sorts ultimately degrade system performance and because tagalong sorts ultimately improve system performance, if disk space is no problem, you should use tagalong sorts. If disk space is a problem, consider purchasing more disk. The amount of time saved by using tagalong sorts instead of addrout sorts probably will pay for the new disk drive in a hurry.

Using #GSORT vs. Alternate Indexes

answered by Ron Mendel

Q We've been looking for a way to speed up daily report processing in our S/36 shop. In particular, we'd like to reduce the time our applications spend sorting files with the #GSORT utility. Is there another sorting method that doesn't take so long?

A When your report requires you to process a file in an order other than the physical record order, consider using the BLDINDEX utility procedure. My tests indicate that BLDINDEX is up to four times faster than #GSORT — with best performance obtained when you give BLDINDEX a 64 K region on a machine that is not swapping heavily. By processing the file via an alternate index (built by BLDINDEX), your report application will perform considerably faster. Be aware, however, that BLDINDEX is useful only when your report must process the entire file. If your report selects only a portion of the file for processing, you must use #GSORT because BLDINDEX does not allow selective Include or Omit functions like #GSORT does.

File Output Using DISP-OLD

by Alex Barish

On the S/36, you can specify DISP-OLD (disposition = old) in the FILE OCL statement to indicate that you want to use an existing file as output. This specification amounts to writing over the old data, not to be confused with adding records to an existing file. In any program that creates a new copy of a file (e.g., a transaction file), you can specify DISP-OLD in the output FILE statement rather than use the DELETE procedure to delete the old copy of the file and then use the BLDFILE procedure to create a new (empty) copy. The DISP-OLD specification in a FILE statement resets the number of records to zero, in effect creating an empty copy, and is much faster than a DELETE followed by a BLDFILE.

Also, in a job that uses SORT, you can specify the same file name on the input and output file statements, with DISP-OLD specified in the output file statement, to sort the file in place. If you use this technique, SORT does not create another copy of the input file; it simply rearranges the records within the existing file. Use this approach with caution (i.e., have a current backup copy of the file) because if any such job fails to run to completion for any reason (e.g., power failure), you may lose the data in the file.

File Extends Explained

answered by Mike Patton and Gary Kratzer

Q Please explain what happens when an EXTEND is executed on a file. Does each extend relocate a file to a portion of the disk large enough to handle the size of the file plus the extend value? I'm hoping your explanation will help me understand the following scenario. The sequence

System: S/36, 90 MB, Release 4
Available disk space: 4,500 blocks
File(s) being extended: 2,600 blocks
Extend value: 200 records
Number of records available before extend: 30
Number of records to be added to file: 250

causes the file to be extended more than once. Disk space is minimal, and I've run a compress right before the program that adds the records. If the file is extended more than once when I run the program, a message is issued that says the file is full. I increase the extend size to prevent multiple extends, which solves the problem. But why?

A When a file is extended, it is copied to another place on disk that has been allocated storage based on the size of the original file plus the extend value (unless the file is not indexed or is an alternate index and enough space is available immediately after the file to accommodate the extend). The original space the file occupied then is made available. The number of records available after an extend generally is larger than the requested number because file allocations are rounded up to the nearest block.

The following pseudocode illustrates the sequence of events that occurs when a file is extended:

```
Is the file non-indexed or an alternate index?
If so
    Is additional space available immediately after the file?
        If so
            Extend the file by moving end of file pointer (extend in place).
        Else
            Is a larger contiguous area of the disk available for the file?
            If so
                Relocate the file to that area and free original file space.
            Else
                Give file full message
```

Else
    Is a larger contiguous area of the disk available for the file?
    If so
        Relocate the file to that area and free original file space.
        Extend all related alternates.
    Else
    Give file full message.

In your case, the file probably was extended once, which used up most of the available disk space. Then, when a second extend was attempted, no room was available for the new file — hence, the "file full" message. Forcing only one extend cured the problem because there was plenty of contiguous space before the first extend was executed.

File Extends and EDF-Wait

answered by Gary Kratzer and Mike Patton

Q On several occasions, I have encountered a status of "EDF-Wait" when displaying the Status Users screen on our S/36 Model 5364. I cannot find any reference to this condition in the IBM-supplied documentation. What causes this condition? How severe is it? How can it be avoided?

A An EDF-Wait can occur on all S/36 models. When this message appears on the Status Users screen, it indicates that, in the current program, a file being output or added to has filled up. The file is being extended automatically by a value that is either an attribute of the file (i.e., the EXTEND parameter established when the file was created) or that has been specified by the OCL in the procedure that is running.

This is not a "severe" condition unless you have too little contiguous disk space to allow the file to EXTEND by the value specified, in which case the program will fail with an error message and a difficult recovery effort may be necessary. EDF-Wait can be indicative of a larger problem (i.e., that your EXTEND value is so small that many EXTENDs are executed during a given run of the program, thereby reducing overall system efficiency).

Extends cannot always be avoided because it usually is not known how many records will occupy a file. Extends can be reduced, though, by specifying a larger EXTEND value. You can specify a larger value by putting an extend value on the // FILE statement or by giving the file a default extend value when it is built via BLDFILE. Note that the EXTEND value on the // FILE statement overrides any default extend value.

Reducing File Extends

by Donald J. Kott

Do you have files that keep getting extended and contain a large number of unused records after they have been organized? Figure 8-13 shows a technique I use to eliminate this problem. First, I use the COPYDATA procedure to organize the file and copy it to file STKORG14. Then, I delete the original file and use the EVALUATE statement to add a fixed number of records to the actual number of records used. Finally, I use the COPYDATA procedure to copy STKORG14 back to the original file name, using the value from the EVALUATE statement in the records parameter of the COPYDATA procedure. When I am finished, the original file size has been incremented by a fixed number of records to allow new records to be added. This technique works well with any file that has an extend value, and the file need never contain a large number of unused records.

Figure 8-13

Technique to eliminate unused records

```
COPYDATA FILENAME,,STKORG14,,,,,REORG,OMIT,2,EQ,'D'
*
DELETE FILENAME,F1
*
// EVALUATE P10=1500+?F'A,STKORG14'?
*
COPYDATA STKORG14,,FILENAME,RECORDS,?10?,,T,NOREORG
*
DELETE STKORG14,F1
```

Calculating File Extend Values

by Nasser Shukayr

When you write a batch OCL procedure to add transactions to a master file, you usually assign the master file a reasonable EXTEND value. Calculating the ideal EXTEND value helps ensure that the file is extended only once and that zero disk space is wasted.

To calculate the ideal EXTEND value, let X be the number of allocated records in the master file, let Y be the number of actual records in the master file, and let Z be the number of records in the transaction file. The ideal extend value for the master file is simply Y plus Z, minus X.

Resizing Files

by Marcia Dore

Because most of my S/36 procedures are coded with EXTEND parameters on the FILE statements to prevent my files from filling up, I usually end the week with files over-allocated, wasting precious disk space.

To alleviate this problem, I use the following technique weekly to resize the files:

```
// EVALUATE P1=?F'A,OLDFILE'?+value
```

where *value* is the average number of records added to OLDFILE during a week.

Then

```
// COPYDATA OLDFILE,,NEWFILE,RECORDS,?1?,,T,REORG
// DELETE OLDFILE,F1
// RENAME NEWFILE,OLDFILE
```

resizes the file accordingly.

I've recovered about 10,000 blocks of disk space using this technique, and, as an added bonus, this technique holds down the number of EXTENDs required during processing.

Clearing Test Files

by Ron Mendel

Code on diskette:
Procedure UTERASE

One of the most tedious chores in testing S/36 programs is deleting test files, tracking down the attributes, and building a new (empty) file for the next round of testing (which is necessary because the programs you are testing "expect" the output files to be empty). Before you can delete and rebuild test files, you must identify the test file's attributes by reviewing a VTOC listing. I have alleviated this problem with the S/36 utility UTERASE (Figure 8-14). Utility UTERASE prompts for a file name and then invokes $COPY to copy a test file to a temporary work file. The // SELECT RECORD,FROM-0,TO-0 statement allows the utility to copy all file attributes, but does not copy records. The DELETE and RENAME statements then ensure that you end up with a new file that has the same name as the old file.

Figure 8-14
Procedure UTERASE

```
* Procedure     UTERASE
* Function   .  Erases all records in a file
* Parameters..  1  Name of file to erase
// IF  ?1?/ * 'Enter the label of the file to erase (or press Enter to exit)'
// IF  ?1R?/         RETURN
// IFF DATAF1-?1?    RETURN
// IF  DATAF1-UTERASE DELETE UTERASE,F1
*
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-?1?
// FILE NAME-COPYO,LABEL-UTERASE
// RUN
// COPYFILE OUTPUT-DISK
```

```
// SELECT RECORD,FROM-0,TO-0
// END
*
// DELETE ?1?,F1
// RENAME UTERASE,?1?
```

Creating Empty Test Files

by David C. Schlosser

Code on diskette:
Procedure CRTEFL

On the S/36, when testing a revised program, you usually need to create copies of existing files so the testing does not disturb live data. Creating copies of existing files requires using the COPYDATA procedure to make copies of master files and using the BLDFILE procedure to create empty transaction files. The problem is that you must find the record length, key length, and other information in the program listings and enter that information into a procedure to build each empty transaction file.

However, if you use the $COPY utility, you can create an empty transaction file without knowing its "vital statistics." Figure 8-15 shows the necessary OCL statements. In the figure, *nnnn* is the number of records to be allocated to the test file. The key to this technique is the line

```
// SELECT RECORDS,FROM-0,TO-0
```

which keeps the $COPY utility from transferring any records into the new file.

Figure 8-15

Procedure CRTEFL

```
* Procedure      CRTEFL
* Parameters     1  Copy file name
*                2  Test file name
*                3  No  of test file records
// IF  ?1?= * 'Enter the name of the file to copy (or press Enter to exit)'
// IF  ?1R?=       RETURN
// IF  DATAF1-?1?  GOTO TNAME
// PAUSE 'File ?1? not found'
// RETURN
*
// TAG TNAME
// IF  ?2?= * 'Enter the name of the test file (or press Enter to exit)'
// IF  ?2R?=       RETURN
// IFF DATAF1-?2?  GOTO COUNT
// * 'Test file ?2? already exists   Delete it? (Y=Yes, N or Enter=exit)'
// IF  ?4R?=       RETURN
// IFF ?4?='Y'     RETURN
DELETE ?2?,F1
*
// TAG COUNT
// IF  ?3?= * 'Enter the number of test file records (or press Enter to exit)'
// IF  ?3R?=       RETURN
// IF  ?3?=0000    RETURN
*
// LOAD $COPY
// FILE NAME-COPYIN,DISP-SHR,LABEL-?1?
// FILE NAME-COPYO,RECORDS-?3?,LABEL-?2?
// RUN
// COPYFILE
// SELECT RECORD,FROM-0,TO-0
// END
```

Dump Files Explained

answered by Mike Patton

Q On the S/36 VTOC listing, I notice something called #DUMP.nn, where *nn* equals a number from 00 to 99. I have three of them on the VTOC, but I can't find any information about them in the system reference manual. What are they, and where do they come from?

A "Dump" files are created when the system recognizes a program or hardware error that makes it impossible (or dangerous) to continue with the task running at the time. The system's response is, quite simply, to end abnormally, thus placing most of the contents of main and control storage into the file #DUMP.nn. The information in this file can help the savvy user (or IBM, in instances where someone discovers a problem with the SSP) determine the cause of the system failure — and, with luck, solve the problem.

However, one of two situations appears to be occurring on your system: (a) one or more of your users is running flawed program(s) and is responding to the system error message without determining (or worrying about) the reason, or (b) the flawed program is set up to use the autoresponse facility, thereby relieving your users of the necessity to respond to the error(s). In either case, it is important to determine which of your programs is failing. Main storage dumps do tend to get in the way of productive computer usage.

Calculating Indexed File Size

answered by Ron Elliott

Q I am puzzled by a data-storage calculation reported by our S/36. An indexed file on disk is allocated for 500 records, with a record length of 64 bytes. By subtracting the beginning location of this file from the beginning location of the next file, the file in question clearly occupies 16 blocks. At 40 records per block, I figure there should be enough space in the file for 640 records, but the VTOC says it will hold only 512 records. What gives?

A In indexed files on the S/36, a relatively small amount of file space is used to store the file index, which occupies disk space immediately preceding the data area of the file. One index entry exists for each record in the file, and each entry in the index occupies a number of bytes equal to the key length plus three. In the example you cite, 500 64-byte records occupy 125 sectors (12.5 blocks) of disk space (at 256 bytes per sector), and the index occupies the other 35 sectors (3.5 blocks) in the same file.

To calculate the number of sectors required for the index of a file, you can follow a simple three-step procedure. First, compute the index entry length by adding three to the key length. Second, divide 256 (the number of bytes in a sector) by the result of step 1, discarding any remainder. And third, divide the total number of entries (i.e., records in the file) by the result of step 2. The rounded result will be the number of sectors required for the index.

Processing Indexed Files vs. Sequential Files with Alternate Indexes

answered by Mel Beckman and Mike Patton

Q Which is better as the primary file on the S/36, a traditional indexed file or a sequential file with an alternate index? If two programs are processing two identical files, is there any significant additional system overhead associated with alternate index processing?

A Despite slight additional overhead, the alternate index is the best way to handle indexed files on the S/36. With alternate indexes, you gain global key update ability. You also can put the parent file on a different drive from the index and thereby improve performance. To reorganize the file, you simply read through the index best suited for the physical order of the particular job being processed.

If the file must be reorganized, the "file" on which the COPYDATA procedure or $COPY program should run is the alternate index whose order, by the definition of its key, most closely approximates the order in which the file most frequently needs to be accessed. This rule holds true, regardless of whether multiple keys are defined for the file. A REORG would be specified, but the output file would be S (sequential), and it would be in order physically by the key field(s) specified in its index. The closer the relationship between a file's most frequently used key order and its physical sequence, the faster it can be processed.

After the file has been reorganized, the "disorganized" copy of the file must be deleted. But before deleting the disorganized copy, you must delete the file's alternate indexes. Once everything is deleted, the organized copy of the file may be renamed to the name of the original, and its alternate indexes may then be rebuilt.

Processing Large Indexed Files

answered by Mike Patton, Mel Beckman, and Barry W. Knapp

Q We have a *huge* indexed file on our S/36 (500,000 records) to which we add 10,000 to 30,000 records a day. This file is empty at the start of the month and full at the end of the month. As the month progresses, the job that adds records to this file takes longer and longer to finish. By the end of the month, the job takes forever to run! Is there a way to speed the process of adding records to this file?

A There may be two reasons why your S/36 takes so much time to add records to your large indexed file: duplicate key testing or a large index overflow.

In duplicate key testing, when your program attempts to add a record to the file, disk data management on the S/36 must scan the entire index (as well as the index overflow area that contains the keys for records added to the file since the last key reorganization) to see whether the key already exists in the file. As more records are added to the file, the S/36 takes longer and longer to check the index for a duplicate key.

If duplicate keys are not a concern of yours (i.e., if you know that no duplicate keys will ever exist in the file, or if you don't care if they do), you can use the BYPASS-YES parameter on the output file's // FILE statement to dramatically speed the process of adding lots of records to the large indexed file. Specifying YES for this parameter instructs the S/36 not to check the index area for a possible duplicate key, so a record is added directly to the file.

If you use BYPASS-YES, the job that adds records to your large indexed file should perform more consistently. In other words, at the end of the month, it won't take very much longer to add records to the file than it did at the beginning of the month (the difference will be the time required to extend the file if your file is extendable). For more information about the BYPASS-YES parameter, see Chapter 5 of the *S/36 System Reference Manual* (SC21-9020).

Another reason it takes so much time to add records may be a large overflow. An overflow is that portion of the index containing keys added since the last full keysort. The prime portion of an index is the portion that contains all the sorted keys upon completion of a full keysort. For keysort performance reasons, keysort does not always remove the overflow (i.e., it does not always merge the overflow with the prime).

An index with no overflow or an index with a small overflow will help performance in duplicate key processing as well as add or update key processing. This is due to two reasons:

1. There is a storage index on only the prime portion of the index.
2. The overlow area is maintained by the SSP in sorted order. The time it takes to add a key to the index increases as the size of the overflow increases.

One way to remove the overflow is to run the keysort procedure and specify the CHKDUP (check for duplicate keys) parameter. If your file does have duplicate keys, you will get a SYS-1367 (take a 0). To prevent these halts from stopping you on an overnight run, code your autoresponse accordingly. If this type of keysort does improve performance, you may want to schedule such keysorts regularly.

Keeping Large Indexed Files Open

by Gary T. Kratzer and Nasser Shukayr

Code on diskette:
Procedure KEEPOPEN
RPG program KPOPEN

In the struggle to keep S/36 interactive response times acceptable to users, programmers must pull rabbits out of hats. One surefire trick to improve interactive response time is to improve program initiation time. The biggest culprit of slow program initiation is the indexed file because the system must scan the entire index to build a storage index. And when storage indexes are built over and over again for the same large indexed files throughout the day, there can be a significant cost in interactive initiation time. But you can rectify this situation by keeping all frequently accessed indexed files open throughout the day.

When large indexed files are kept open, their storage indexes are built only once, and all programs using that file share the same storage index. In other words, each user does not "own" his or her own storage index. Only the first user of the file must endure the initiation delay caused by the storage index being built.

To keep indexed files open, you must run a program that remains active continuously. A MRT-NEP program — a Multiple Requester Terminal program that has the Never-Ending-Program attribute set — serves this purpose well. Unlike an ordinary Single-Requester-Terminal (SRT) program, which is not capable of releasing the requesting display station, a MRT-NEP can release its requesters and remain active. Although there are other methods you can use to keep files open, the MRT-NEP program offers an advantage that many other methods do not: the MRT-NEP can be canceled by a workstation other than the system console. In addition, the MRT-NEP does not tie up a workstation or cause the system to perform unnecessary processing because once activated, it remains in a suspended state.

Let's look at the basic components of the MRT procedure and program needed to keep large indexed files open. Following the MRT procedure name, which is KEEPOPEN (Figure 8-16) in our example, you can key data to be passed to the MRT program as a workstation input record. For example, if you key in KEEPOPEN NOW IS THE TIME, the characters NOW IS THE TIME are passed to the program as the first input record. When you key the name of the MRT procedure, the succeeding characters are saved until the MRT program performs its first input operation. In our example program, KPOPEN (Figure 8-17), the first input record is used to pass a cancellation code so you can terminate the program on demand. The scheme is simple: a blank input record starts the program, and a nonblank input record cancels the program. The input record is blank if you don't key any data following the MRT procedure name.

File Statements and Specifications

Coding, at least of file names, is installation-specific. You can, however, get a good idea of what the file statements should look like by using our examples. When you create procedure KEEPOPEN, you must answer yes to the "MRT procedure?" prompt before replacing the procedure in the library to let it invoke MRT program KPOPEN. The system checks only the procedure name to see whether the MRT program is already active; it does not check to see which library the procedure comes from, only that it's active, so make sure the procedure name is unique within the entire system.

Because the program that accesses the files to be kept open is input only, you define the disposition of the file as SHRRM, which specifies that the MRT program may only read the file but that all other programs may read or modify the file.

You can override the SSP default storage index size by specifying the maximum size of the storage index to be built with the STORINDX keyword on the // FILE statement. Before doing this, you should understand how the SSP calculates the storage index size. If your system has 128 K of main storage, the maximum default storage index size is 2 K. On systems with more than 128 K of main storage, the maximum default storage index size is 8 K. The SSP uses a combination of factors to determine the size of the storage index. You can override this SSP-computed size by specifying a larger size for the storage index, which often speeds up indexed reads. However, depending on the file's key length and number of records, there may be a maximum storage index size that the SSP can use effectively. Chapter 8 in IBM's *Concepts and Programmer's Guide* (SC21-9019) explains how to compute an efficient storage index size.

If you want to build a storage index for referenced files only, do not specify STORINDX-YES on the // FILE statement. If you want to build storage indexes for an entire family of files by referencing just the parent, do specify STORINDX-YES on the // FILE statement.

You can reference up to 15 randomly processed files in a single RPG program. If you want to keep more than 15 large indexed files open, you must create more than one RPG program and MRT procedure. You could use a different program or procedure to keep open groups of large files related to the same application. Remember that a file with alternate indexes requires only one file statement in the RPG program (i.e., referencing the parent file causes the creation of a storage index for each alternate index defined over the parent file).

The primary file in program KPOPEN is a WORKSTN file. The program does not read or write to the workstation file; all input for the workstation program actually comes from data passed through the first input record by the MRT procedure. The program always processes exactly one input record and releases the requester after handling this input record. Because the program never reads or writes to the workstation device, you don't need to define a screen format member; thus, in the F-specs, you code a KFMTS continuation line specifying *NONE.

For the workstation file record length, specify at least as many characters as the maximum amount of input data you expect to use to cancel the program. For example, if you want to call (and thereby cancel) the program with the passed data of CANCEL, specify a record length of at least six. If you key in more characters than the record length allows, a blank input record is passed to the program.

I-Specs

For the workstation file, you must define two record types in the I-specs: a blank record (ignored by the program) to start the program and a nonblank record to cancel the program. Remember, the input record does not actually come from the workstation device; the record comes from the data that is keyed in after the MRT procedure name.

C-Specs

To avoid terminal errors, you must reference each disk file defined in the F-specs in a CHAIN or READ operation in the C-specs. Although the operations are never executed, you must code at least one input operation for each chained, demand, or full procedural input file in your program; otherwise, the RPG compiler issues an error message. You also need to provide a way to signal the program to go to end-of-job when the cancellation record is received, which can be done by setting on LR when the input indicator for a nonblank record is on.

O-Specs

In the O-specs, you code an R (release) in column 16 of program KPOPEN to release the display station when a record other than the cancellation record (i.e., the nonblank record) is processed. No screen format name is required in the O-specs.

After you have coded the RPG specifications, you must specify how you want the program compiled. On the RPG compiler (RPGC) procedure, specify 1 as the value for the "Maximum number of requesting display stations" parameter. Also, specify NEP as the value for the "Never-Ending-Program" parameter.

Now that you have created program KPOPEN, the best way to use it is to call it conditionally from an existing procedure just before a large indexed file is used. You also can call it at the start of the day; you may want to include it in one of your initial startup procedures.

You cancel program KPOPEN by keying KEEPOPEN with a nonblank first parameter. The parameter is passed to program KPOPEN as an input record; program KPOPEN is coded so a nonblank input record causes indicator LR to be set on, which cancels the program (Figure 8-18). Another way to cancel program KPOPEN is to use the STOP SYSTEM command, which causes an end-of-program status for all MRT-NEP programs as soon as the last requester is released. Because program KPOPEN normally has zero requesters, the STOP SYSTEM cancels the program immediately.

Because all existing references to the data file (e.g., any indexed data files you choose to open and keep open using this technique) must allow file sharing, you may have to make a few changes to existing procedures. If you have files that can't be shared, you can either modify the existing FILE statement to allow file sharing, or, if the application requires a non-shared file, you can add the necessary OCL statements to cancel the KEEPOPEN procedure before running the application. If you have many programs that do not allow file sharing for large indexed files, you may need to make a lot of changes to your FILE OCL statements. But the performance improvement this technique offers is worth the effort.

And now for the bottom line: the results of the benchmarks performed on a dedicated S/36 5360 Model D with a frequently used interactive program that references a large indexed file (630,000 records) along with an alternate index (Figure 8-19). The program also references other smaller files. When not using this technique, it takes about 22 seconds to initiate the interactive program on a dedicated system; if the system is being used by other jobs, the program takes approximately 47 seconds to be initiated.

When using this technique, however, and when the large indexed file and the large alternate index are already open, it takes only about one and one-half seconds to initiate the program on a dedicated system; on a nondedicated system, the initiation time is less than three seconds. If all the indexed files used by the program are already open, initiation time is less than one second on a dedicated system, and initiation time is less than two seconds on a nodedicated system.

Every time you use program KPOPEN, 21 seconds are saved on a dedicated system, and 45 seconds are saved on a typically loaded system. When you multiply the number of large indexed files in the VTOC by the number of times you use the inquiry program each day, it adds up to significant time savings achieved with relatively little programming effort.

No single technique can guarantee that your system response time will change suddenly from unacceptable to acceptable. However, you can reduce significantly the amount of time your system spends doing nonproductive work by speeding up the running of your large indexed files. So go for it! Write a simple MRT-NEP program, and start initiating your programs faster.

Figure 8-16

MRT procedure KEEPOPEN

```
// LOAD KPOPEN
// FILE NAME-APTRANS,DISP-SHRRM
// FILE NAME-APVEND,DISP-SHRRM
// FILE NAME-CUMASTER,DISP-SHRRM
// RUN
```

Figure 8-17

Program KPOPEN

```
*.. . 1 ... 2 ... 3 .. 4 . .5 . ..6 . . 7 . . 8
     H                                                                   KPOPEN
     F*************************************************************
     F*                                                           *
     F*    Program name   KPOPEN  (a MRT-NEP program)             *
     F*    Date   01-25-88                                        *
     F*    Purpose:  This is a MRT-NEP program that will keep the *
     F*              large indexed files open                     *
     F*    Special instructions   Compile with Number of Requestors=1, *
     F*                           and with the NEP option set     *
     F*    Files used:  SCREEN   workstation file                 *
     F*                 APTRANS  accounts payable transaction file *
     F*                 APVEND   accounts payable vendor file     *
     F*                 CUMASTER customer master file             *
     F*                                                           *
     F*************************************************************
     F* +---------------------------------------------------------+
     F* |                                                         |
     F* |               I N D I C A T O R   U S A G E             |
     F* |                                                         |
     F* |   01  Blank input record, used when starting the program |
     F* |   02  Non-blank input record, used to cancel the program |
     F* |                                                         |
     F* +---------------------------------------------------------+
     FSCREEN  CP  F  80  80            WORKSTN
     F                                                  KFMTS   *NONE
     FAPVEND  IC  F 120 120R 5AI      1 DISK
     FCUMASTERIC  F 384 384R 6AI      1 DISK
     FAPTRANS IC  F  72  72R 7AI      1 DISK
     ISCREEN  NS  01   1 C
     I        OR  02
     I*
     IAPVEND  NS
     I                                         1   5 KVEND
     ICUMASTERNS
     I                                         1   6 KCUST
     IAPTRANS NS
     I                                         1   7 KTRANS
     C* +---------------------------------------------------------+
     C* | A non-blank input record causes the program to be cancelled.|
     C* +---------------------------------------------------------+
     C         02          SETON                     LR
     C* +---------------------------------------------------------+
     C* | The following instructions are never executed   The RPG |
     C* | compiler requires at least one input operation for each |
     C* | input file                                              |
     C* +---------------------------------------------------------+
     C        02NLR        DO                                  <Never executed>
     C             KVEND     CHAINAPVEND              02
     C             KCUST     CHAINCUMASTER            02
     C             KTRANS    CHAINAPTRANS             02
     C                       END
     O* +---------------------------------------------------------+
     O* | Release the requesting workstation                      |
     O* +---------------------------------------------------------+
     OSCREEN  DR       01
```

Figure 8-18

```
// IF ACTIVE-KEEPOPEN  KEEPOPEN CANCEL
COMPRESS
```

Calling KEEPOPEN with the cancel option

Figure 8-19

Some actual timing data, elapsed time between requesting an actual interactive program and the appearance of the first screen format

Conditions	Dedicated system	Actual intra-day timings
No storage indexes in memory	22 seconds	47 seconds
Indexes for the two large files (>630,000 records) in memory	< 2 seconds	< 3 seconds
Indexes in memory for all files used by the program	< 1 second	< 2 seconds

Processing Alternate Indexes in COBOL

answered by Georgia Agallianos

Q Our shop (with a S/36 5362) is one of the few that use COBOL. I have read about alternate index processing in the COBOL manual, but I'm still not clear about how to do it. Where can I find information and examples on alternate index processing?

A Generally, alternate index files are not treated any differently by COBOL (or any other high-level language) than normal indexed files are. You simply specify the file as indexed, but you specify the same key as that defined in the alternate index — not as it is defined in the physical "parent" file. Next, you use the name of the alternate index to code the // FILE statement. As with normal indexed files, COBOL expects you to specify whether the file is duplicate-capable within the program. If you define your physical file so it has unique keys, use alternate indexes to specify duplicate keyed paths. COBOL will halt with a runtime error when the file is opened unless you have informed it to expect the duplicate key capability of the file. Chapter 8 in the *S/36 Concepts and Programmer's Guide* (SC21-9019-5) contains additional information.

Keysorting During IPL

answered by Mel Beckman and Gary T. Kratzer

Q During IPL, my S/36 displays the message that one of our large files is being keysorted. If I immediately IPL the system again after the first IPL is finished, the same file is keysorted again, even though no records have been added to the file. Was the file really keysorted during the first IPL?

A Under SSP Release 4.0, IBM issued the message "Conditionally sorting keys for file xxx" during IPL to indicate that a keysort might be in progress for a particular file. By comparing the number of records in the overflow index with the total number of records in the file, the SSP then determined whether the file really needed keysorting. If the percentage of overflow records exceeded a certain threshold (about 7 percent), the file was keysorted. For large indexed files, keysorting did not occur until quite a few records had been added to the file. At SSP Release 5.0, IBM removed the word "conditionally" from the IPL keysort message, leading you to believe that the file was actually being keysorted when it really wasn't.

Keysorting a large file may improve system performance, even if the SSP doesn't think it's necessary. You can force a keysort by running the KEYSORT procedure with the CHKDUP parameter. If the file contains duplicate keys (even though duplicates are allowed), you will receive duplicate key messages that you easily can bypass by responding with a 1 to the second message. The file still will be keysorted.

Blocking Records

answered by Mel Beckman

Q I have some S/36 disk data management questions about blocking files. I have a basic working knowledge of DDM; for example, I know that read and write operations are performed in 256-byte increments, and so on. For the sake of the discussion below, let's not consider using CACHE.

1. Presume that the file in Figure 8-20a is in key sequence and that the key to the file is the RRN. If an initial

```
'0001'    SETLL FILE
          READ FILE
```

is performed, are the first four records of this file in the program's buffer because DDM I/O is always in 256-byte increments, even though the F-spec doesn't block to 256?

2. The file in Figure 8-20b is the same as the file in Figure 8-20a except this file is blocked to 512 bytes. If the following instructions

```
'0003'      SETLLFILE
            READ FILE
```

are performed, are RRNs 1 through 8 or RRNs 3 through 10 put into the program's buffer?

3. Next, the instructions

```
'0005'      SETLLFILE
            READ FILE
```

are performed. Does another physical disk I/O operation take place here? If it does, what records are in the buffer?

4. As long as a requested record is already in the program's buffer, will physical disk I/O not be performed? Will the answers to these questions change if the file is defined as an update file? Does the file's OCL DISP-parameter affect any of this? I would imagine that if program A's buffer holds RRNs 1 through 8, and another program (B) updates RRN 6, when program A goes to read RRN 6, DDM will know to reread those records (will just the sector that holds RRN 6 be reread)?

A 1. Yes, four records are read because of the 256-byte disk sector size (and the record size you give divides evenly into 256).

2. Records 1 through 8 go into the buffer because the buffer must begin with the first 256-byte sector that contains the requested record. In this case, record 0003 is the second to last record in the first sector, so records 1 and 2 are along for the ride.

3. No, when you do a subsequent read of record 0005, no more physical I/O occurs because the record is in the buffer already.

4. As long as the record exists in the buffer, no physical I/O occurs on either input or update, unless the file is shared. For shared files, if another program updates a record that's currently in your buffer, DDM invalidates your buffer (i.e., makes it look empty). Thus, your next disk request results in an automatic reread of the entire buffer, which gives you a new copy of the record.

Figure 8-20a

File blocked to 256-byte increments

```
*       1 .      2      . 3       4        5        6        7
    FFILE    IF  F  64  64L 4AI      1 DISK
```

Figure 8-20b

File blocked to 512-byte increments

```
*    . . 1 . .   . . 2      . 3 . .    4  . .    5   .    6      . 7
     FFILE   IF  F 512  64L 4AI     1 DISK
```

Running a Dedicated COPYDATA

by Donald J. Kott

Code on diskette:
Procedure STKORG14

If you use the COPYDATA procedure to remove deleted records, problems can occur because COPYDATA can copy a file that another job is reading. Figure 8-21 shows a procedure in which I use a // FILE statement before a // COPYDATA statement to prevent COPYDATA and other jobs from interfering with each other.

The WAIT-NO parameter of the FILE statement lets procedure STKORG14 continue even if another job is using the designated file. Immediately after the FILE statement, the procedure checks the return code. A return code of 2030 or 2031 indicates the file was not acquired because it was in use. If the file is in use, the operator receives a message that procedure STKORG14 cannot be run, and the job is canceled. A return code of 0000 indicates that the file was acquired.

If the file is acquired, the DISP-OLD and JOB-YES parameters in the FILE statement prevent other users from reading the file until the COPYDATA procedure has completed the reorganization of the file. With the file free—and free from interference—procedure STKORG14 reorganizes the file and removes deleted records. It then deletes the designated file and renames file STKORG14 to the original file name.

Figure 8-21

Procedure STKORG14. Substitute your own name for FILENAME and the work file STKORG14, and tailor the COPYDATA statement for your file.

```
// FILE NAME-FILENAME,JOB-YES,DISP-OLD,WAIT-NO
* Check to see if file is being used
// IF ?CD?-0000 GOTO OK
// * 'The file is being used, cannot organize'
// PAUSE 'Program Canceled, Enter <0> to continue'
// CANCEL
// TAG OK
*
// IF DATAF1-STKORG14  DELETE STKORG14,F1
*
COPYDATA FILENAME,,STKORG14,,,,,REORG,OMIT,2,EQ,'D'
*
DELETE FILENAME,F1
*
RENAME STKORG14,FILENAME
```

Reorganizing Files Automatically

by Perry Gardai

program by Steve Leichman

Code on diskette:
Procedures REORG, REORG1
RPG program REORG

Looking for an easier way to reorganize your index files? This S/36 utility automatically deletes alternate indexes, creates reorganization OCL, performs the actual reorganization, then rebuilds the alternate indexes after reorganization is complete — all with one easy command.

File reorganization: that painful, necessary process that you really should perform regularly on your indexed files, but only get around to sporadically. It's painful because you're always forgetting about those alternate indexes that must be deleted before a reorganization. It's painful also because you have to enter the same long list of parameters over and over again in the COPYDATA statement if you want to reorganize multiple files. But regular file reorganization is necessary because files in key-sequential order, with delete-coded records removed, are better stewards of response time and disk space than disordered files that store many "deleted" records. And so you may welcome REORG, a S/36 utility that automatically:

- deletes all alternate indexes associated with a file
- retrieves reorganization parameters from system data
- reorganizes the file in primary key-sequential order, dropping deleted records
- re-creates all alternate indexes over the file.

The REORG utility consists of three procedures — REORG, REORG1, and REORG2 — and one program — also named REORG. Procedure REORG is the master procedure of the utility and should be stored in the system library or in your tool kit library.

Procedure REORG (Figure 8-22) first checks for parameter 1, the name of the file to be reorganized. The procedure performs a CATALOG to retrieve information about the file and its alternate indexes and then saves that information in a disk file for input to program REORG to create OCL that performs the major functions of the utility.

Next, procedure REORG calls program REORG (Figure 8-23), which creates the OCL required to specify the appropriate parameters, delete the alternate indexes, perform the actual reorganization, and rebuild the alternate indexes that may be attached to parent files. Procedure REORG then copies the OCL (created by program REORG) to a library procedure member named REORG2. Note that the OCL will be different for every execution of the utility because the OCL is designed to delete and rebuild specific alternate indexes.

Finally, procedure REORG calls procedure REORG2 (Figure 8-24), which deletes the alternate indexes and calls procedure REORG1 (Figure

8-25) to perform the actual parent file reorganization. After the reorganization step, control returns to procedure REORG2, which rebuilds all alternate indexes for the parent file. Note that the position of the key in relation to the file is established by the three // POSITION statements. The order of the POSITION statements is critical because it ranks the noncontiguous fields in order of significance. After the alternate indexes are rebuilt, the utility ends, and you may begin the next reorganization.

Note that you are responsible for ensuring that the parent file, and all alternate indexes defined over it, are not in use by any other jobs because the S/36 will not allow the $DELET or COPYDATA (with the delete parameter) statements to be executed if the files are in use. You also should be aware that if your S/36 allows more than one job from the job queue to be executed concurrently, you should not run this utility if someone else is running it (e.g., a user at another workstation or a previous submission on the job queue).

This simple utility can help any S/36 shop reorganize data files without the worry about forgotten alternate indexes and without the need to key repetitive parameters in COPYDATA statements. The time you save will certainly be worth the effort.

Figure 8-22

Procedure REORG

```
// IFF ?1?/  * '?1? is being reorganized.'
// IF ?1?/  * 'This will REORGANIZE a data file '
// IF ?1?/  * 'KEY IN THE FILE TO REORGANIZE.'
// IF ?1R?/  CANCEL
// IFF DATAF1-?1R?  * '?1? IS NOT ON DISK'
// IFF DATAF1-?1R?  PAUSE
// IFF DATAF1-?1R?  RESET REORG
**   CATALOG THE DISK AND STORE IN DISK FILE
// IF DATAF1-REOR?WS?X1  DELETE REOR?WS?X1,F1
// LOAD $LABEL
// RUN
// DISPLAY LABEL-?1?,UNIT-F1,OUTPUT-REOR?WS?X1
// END
// IF DATAF1-REOR?WS?X2  DELETE REOR?WS?X2,F1
**   SORT THE CATALOG
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-REOR?WS?X1,RETAIN-S
// FILE NAME-OUTPUT,LABEL-REOR?WS?X2,RECORDS-?F'A,REOR?WS?X1'?
// RUN
     HSORTR     1A        3X 132
     O C   1   5EQCUSER
     OOC   1   5EQC*****
     I C  22  22EQCD
     IOC  22  22EQCI
     IOC  22  22EQCS
     IOC  22  22EQCX
     IOC  77  77GECO
     FNC  23  23                   DUMMY
     FDC   1 132                   ALL DATA
// END
// IF DATAF1-REOR?WS?X3  DELETE REOR?WS?X3,F1
**  READ THE CATALOG AND SET UP OCL
// SWITCH 10000000
// LOAD REORG
// FILE NAME-INPUT,LABEL-REOR?WS?X2,RETAIN-S
// FILE NAME-REORG2,LABEL-REOR?WS?X3,RECORDS-500,EXTEND-100
// RUN
// SWITCH 00000000
**  PLACE THE OCL INTO SESSION LIBRARY
```

```
// LOAD $MAINT
// FILE NAME-REOR?WS?X3,RETAIN-S
// RUN
// COPY TO-?SLIB?,FROM-DISK,FILE-REOR?WS?X3,NAME-REORG2,RETAIN-R,LIBRARY-P
// END
REORG2
```

Figure 8-23

Program REORG

```
*... .   1  .  .   2      .  3 .   ..  4      .. 5      .  6         7         8
     H     024                                                          REORG
     FINPUT   IP  F 132 132            DISK
     FREORG2  O   F 120 120            DISK
     F*
     F*   FUNCTION    - REORGANIZE A DATA FILE
     F*              -  U1-REORG   U2-CLRPFM   U3-PURGE ROUTINE
     F*              -  U4-DELETE A FILE
     F*                      GRANAT DATA - SL - 3/86
     F*
     E                    AN         99  8              ALTERNATE NAME
     E                    PO         99  4 0            KEY POSITION
     E                    LE         99  2 0            KEY LENGTH
     E                    NC         99  2              NON-CONTIG FLAG
     E                    ST         99  1              STATUS
     E                    NCC        99  1 0            NON-CONTIG COUNT
     E                    ANW         9  1              NAME WORK ARRAY
     E                    XNW         9  1              NAME WORK ARRAY
     IINPUT   AA  01   22 CX
     I                                         1   8 ALTNAM
     I                                        36  36 STATUS
     I                                        38  38 STATU2
     I                                        70  730POS
     I                                        76  770LEN
     I                                        71  72 NONCON
     I        CC  03   22 C
     I                                        70  730POS
     I                                        76  770LEN
     I        BB  02
     I                                         1   8 FILENM
     C  NO2                GOTO NOTO2
     C* SET-UP THE PARENT NAME
     C                     MOVE *BLANKS   ANW
     C                     MOVEAFILENM    ANW
     C                     Z-ADD1         Z       20
     C            *BLANK   LOKUPANW,Z                   24
     C                     MOVE ','       ANW,Z
     C                     MOVEAANW       PLABEL  9
     C                     GOTO OUT
     C            NOTO2    TAG
     C*
     C                     ADD  1         X       20
     C                     MOVE ALTNAM    AN,X
     C                     MOVE POS       PO,X
     C                     MOVE LEN       LE,X
     C                     MOVE NONCON    NC,X
     C            STATUS   IFEQ '1'
     C                     MOVE STATU2    ST,X
     C                     ELSE
     C                     MOVE STATUS    ST,X
     C                     END
     C   01                Z-ADD1         XX      10
     C   03                ADD  1         XX
     C                     Z-ADDXX        NCC,X
     C            OUT      TAG
     CLR                   EXSR LRSR
     C*
     C* THIS SUBR WILL CREATE THE OCL
     C*
     C            LRSR     BEGSR
     C* GENERATE DELETE RECORDS
     C                     Z-ADD1         X
     C            AN,1     COMP *BLANKS                 23
     C   23                GOTO NODEL
     C            AGN1     TAG
     C            AN,X     COMP *BLANKS                 21
```

```
C   21                GOTO NODEL
C                     SETOF                     26
C           NC,X      COMP 'NC'                     25
C   25      NCC,X     COMP 1                    26
C  N26                EXCPTDELX
C                     SETON                     31
C           X         COMP 99                     2222
C   22                ADD  1         X
C   22                GOTO AGN1
C*
C           NODEL     TAG
C                     EXCPTCOPY
C* GENERATE REBULID RECORDS, IF NOT 'DELETE'
C   U4                GOTO ENDIT
C                     Z-ADD1         X
C           AGN2      TAG
C           AN,X      COMP *BLANKS                  21
C   21                GOTO ENDIT
C           NC,X      COMP 'NC'                     21
C                     SETOF                     414243
C                     SETOF                     4445
C  N21                GOTO NOT21
C* SET-UP THE ALTERNATE NAME
C                     MOVE *BLANKS   XNW
C                     MOVE AN,X      ALTXX    8
C                     MOVEAALTXX     XNW
C                     Z-ADD1         Z
C           *BLANK    LOKUPXNW,Z                    24
C                     MOVE ','       XNW,Z
C                     MOVEAXNW       ALABEL   9
C           NCC,X     COMP 1                        41
C           NCC,X     COMP 2                        42
C           NCC,X     COMP 3                    44  43
C           X         ADD  1         YY      20
C           NCC,YY    COMP NCC,X                45
C           NOT21     TAG
C*
C           ST,X      COMP '2'                      91 DUPES ALLOWED
C                     EXCPTBLDX
C           X         COMP 99                     2222
C   22                ADD  1         X
C   22                GOTO AGN2
C*
C           ENDIT     ENDSR
OREORG2  D        1P
O                                   24 '// COPY LIBRARY-P,NAME-R'
O                                   48 'EORG2                   '
O        E       N31     DELX
O                                   24 '// LOAD $DELET          '
O        E       N31     DELX
O                                   24 '// RUN                  '
O        E                DELX
O                                   24 '// SCRATCH UNIT-F1,LABEL'
O                                   25 '-'
O                         AN,X      33
O        E        31      COPY
O                                   24 '// END                  '
O        E                COPY
O                       U4           6 'DELET2'
O                       U3           6 'CLRPF3'
O                       U2           6 'CLRPF2'
O                       U1           6 'REORG1'
O                         FILENM    15
O        E        31NU4   COPY
O                                   24 '// LOAD $FBLD           '
O        E        31NU4   COPY
O                                   24 '// RUN                  '
O        E       N21      BLDX
O                                   24 '// FILE ATTRIB-X,POSITIO'
O                                   26 'N-'
O                         PO,X      30
O                                   41 ',LENGTH-   ,'
O                         LE,X      40
O                                   48 'DUPKEY-'
O                       91          52 'YES,'
O                      N91          51 'NO,'
```

```
0        E       N21       BLDX
0                                      10 '// PLABEL-'
0                          PLABEL      19
0        E       N21       BLDX
0                                       9 '// LABEL-'
0                          AN,X        17
0        E        21 41    BLDX
0                                      24 '// FILE ATTRIB-X,DUPKEY-'
0                        91            28 'YES,'
0                       N91            27 'NO,'
0        E        21 41    BLDX
0                                      10 '// PLABEL-'
0                          PLABEL      19
0        E        21 41    BLDX
0                                       9 '// LABEL-'
0                          ALABEL      18
0        E        21 42    BLDX
0                                      24 '// POSITIN1-XXXX,LENGTH1'
0                                      25 '-'
0                          PO,X        16
0                          LE,X        27
0                        45            28 ','
0        E        21 43    BLDX
0                                      24 '// POSITIN2-XXXX,LENGTH2'
0                                      25 '-'
0                          PO,X        16
0                          LE,X        27
0                        45            28 ','
0        E        21 44    BLDX
0                                      24 '// POSITIN3-XXXX,LENGTH3'
0                                      25 '-'
0                          PO,X        16
0                          LE,X        27
0        T        31 LRNU4
0                                      24 '// END
0        T        LR
0                                      24 '// CEND
```

Figure 8-24

Procedure REORG2. This shows a sample of a REORG2 procedure created by REORG.

```
// LOAD $DELET
// RUN
// SCRATCH UNIT-F1,LABEL-CKHALT1
// SCRATCH UNIT-F1,LABEL-CKHALT2
// SCRATCH UNIT-F1,LABEL-CKHSL
// END
REORG1 CKHIST
// LOAD $FBLD
// RUN
// FILE ATTRIB-X,POSITION-0013,LENGTH-10,DUPKEY-NO,
// PLABEL-CKHIST,
// LABEL-CKHALT1
// FILE ATTRIB-X,POSITION-0002,LENGTH-27,DUPKEY-NO,
// PLABEL-CKHIST,
// LABEL-CKHALT2
// FILE ATTRIB-X,DUPKEY-YES,
// PLABEL-CKHIST,
// LABEL-CKHSL,
// POSITIN1-0001,LENGTH1-10,
// POSITIN2-0015,LENGTH2-02,
// POSITIN3-0021,LENGTH3-03
// END
```

Figure 8-25

Procedure REORG1

```
** REORG A FILE
// IF ?1R?/  * 'KEY IN THE FILE TO CLEAR OUT '
// IFF DATAF1-?1R?  * '?1? IS NOT ON DISK'
// IFF DATAF1-?1R?  PAUSE
// IFF DATAF1-?1R?  RESET REORG1
// IF DATAF1-CLR?WS?WRK   DELETE CLR?WS?WRK,F1
COPYDATA ?1?,,CLR?WS?WRK,,,,,REORG
DELETE ?1?,F1
RENAME CLR?WS?WRK,?1?
```

Making a File Delete-Capable

by Mel Beckman

Code on diskette:
Procedure SETDEL
Assembler program SETDEL

Those of you who use the S/36 deleted record capability know of an annoying omission by IBM: it is impossible to make a file "delete-capable" after the file has been created. If you inadvertently attempt to delete a record from a file that is not delete-capable, the program is canceled with option 2 or 3. Copying the file with COPYDATA does not get the file into a delete-capable state, and neither does restoring the file from diskette. Seemingly, the only way to get the file into a delete-capable state is by writing a $COPY procedure to copy the file, with DFILE-YES specified in the // FILE statement for the output file. If alternate indexes have been built over the file, you must delete the indexes and rebuild them after copying the file. The RIF (Rochester Irk Factor) increases in direct proportion to the number of records in the file.

Fortunately, a tiny assembler language program can be used to make any file delete-capable. If alternate indexes are associated with the file, they are made delete-capable also.

Program SETDEL is called from procedure SETDEL (Figure 8-26), which can be stored in #LIBRARY. Program SETDEL works on the principle that the SSP reads into memory the VTOC entry for every // FILE statement. Program SETDEL simply sets the delete-capable bit in each VTOC entry, and the SSP automatically writes the entries back to the VTOC at end of job. The SETDEL procedure in Figure 8-26 shows only one FILE statement, but you can code as many as you like — program SETDEL will process them all. Note that you only need to specify a FILE statement for one member in a "family" (parent plus alternate index files) to make the whole family delete-capable.

Figure 8-26
Procedure SETDEL

```
// LOAD SETDEL
// FILE NAME-?1?
// RUN
```

Re-creating Program SETDEL

If you don't have assembler program SETDEL, you can re-create it with procedure MKSETDEL (you don't need IBM's Assembler Language Program Product to install SETDEL).

```
**
** Use #MAINT to copy #CNHLP to SETDEL so we have a load member to patch
**
// LOAD #MAINT
// RUN
// COPY FROM-#LIBRARY,TO-#LIBRARY,LIBRARY-O,NAME-#CNHLP,NEWNAME-SETDEL,RETAIN-R
// END
**
** Patch the new SETDEL member to make it set file delete flags
**
// LOAD $FEFIX
// RUN
HDR
PTF OSETDEL,,#LIBRARY
DATA F2,0000,35A1,184C,75A1,0075,A117,75A1,BA38,A118,40
DATA 40,0010,40F2,812F,75A2,1F36,A218,40F2,811F,70F8,10
DATA 10,0020,10F2,0219,B880,18F2,9003,86A2,79BA,201C,BA
DATA BA,0030,BA80,1086,A288,36A2,1849,F101,1075,A11C,F1
DATA F1,0040,F187,36F4,0004,0400,0000,000B,97FD
END
```

Deleting Multiple Files

by Charles M. Barnard

Code on diskette:
Procedure DEL

Figure 8-27 shows a procedure I wrote to avoid having to write

```
// IF DATAF1-filename DELETE filename,F1
```

every time I wanted to delete a temporary file that might or might not exist. Procedure DEL deletes up to 11 individual files with one call from the terminal.

The procedure performs an existence check before it attempts to delete a file and then deletes the file if the check succeeds. The procedure exits if the passed file name is blank, and the procedure will not let you delete all files.

This procedure may be inserted within most other procedures, as long as the nested depth plus the number of files passed does not exceed 16.

Figure 8-27
Procedure DEL

```
**  DEL MULTIPLE FILE DELETION UTILITY WITH EXISTENCE CHECK   C. BARNARD
**
**  'USAGE  DEL FILE1,FILE2   FILE11   (WILL DELETE UP TO ELEVEN FILES)'
**
** IF JOBQ-NO IF EVOKED-NO * 'DEL ?1?,?2?,?3?,?4?,?5?,?6?,?7?,?8?,?9?,?10?,?11?'
**
// LOAD $DELET
```

```
// RUN
// IFF  ?1?/ IF DATAF1-?1?  SCRATCH UNIT-F1,LABEL-?1?
// IFF  ?2?/ IF DATAF1-?2?  SCRATCH UNIT-F1,LABEL-?2?
// IFF  ?3?/ IF DATAF1-?3?  SCRATCH UNIT-F1,LABEL-?3?
// IFF  ?4?/ IF DATAF1-?4?  SCRATCH UNIT-F1,LABEL-?4?
// IFF  ?5?/ IF DATAF1-?5?  SCRATCH UNIT-F1,LABEL-?5?
// IFF  ?5?/ IF DATAF1-?6?  SCRATCH UNIT-F1,LABEL-?6?
// IFF  ?6?/ IF DATAF1-?7?  SCRATCH UNIT-F1,LABEL-?7?
// IFF  ?7?/ IF DATAF1-?8?  SCRATCH UNIT-F1,LABEL-?8?
// IFF  ?8?/ IF DATAF1-?8?  SCRATCH UNIT-F1,LABEL-?8?
// IFF  ?9?/ IF DATAF1-?9?  SCRATCH UNIT-F1,LABEL-?9?
// IFF  ?10?/  IF DATAF1-?10?  SCRATCH UNIT-F1,LABEL-?10?
// IFF  ?11?/  IF DATAF1-?11?  SCRATCH UNIT-F1,LABEL-?11?
// END
```

Saving History Files

by Thomas Straitwell and Martin Bell

Code on diskette:
Procedure HISTCOPY

When a S/36 is configured to save the history file periodically, the file is copied to a user file called HISTCOPY whenever the system history file becomes 80 percent filled. If the history file fills up twice in the same day, however, the system attempts to create a duplicate copy of the user file HISTCOPY and locks up the workstation until someone intervenes.

You can prevent this situation by using IBM's HISTCOPY procedure. If you add the OCL statements shown in Figure 8-28, the system creates history files named HIST.1, HIST.2, and so on, avoiding duplicate names. Procedure HISTCOPY uses an *n*-positional stack procedure to maintain *n* history files. As the system history file is saved, the *n*th file is deleted, and the others are renamed. We can change the size of the stack easily.

Figure 8-28

Procedure HISTCOPY

```
*
* HISTCOPY PROCEDURE  MAINTAINS AN N-POSITIONAL STACK OF HISTORY
*                     FILE ROLLOVERS
*                           P1  -  N POSITIONS COUNTER
*                           P2  -  N-1 COUNTER
*
* INITIALIZE COUNTERS
// EVALUATE P1,1-5 P2,1-4
* DELETE OLDEST FILE IN STACK
// IF DATAF1-HIST ?1? DELETE HIST ?1?,F1
* RENAME OTHER FILES IN STACK
// TAG LOOP
// IF DATAF1-HIST ?2? RENAME HIST ?2?,HIST ?1?
// EVALUATE P1,1-?1?-1 P2,1-?2?-1
// IF ?1?>1 GOTO LOOP
* RENAME NEWEST FILE
// RENAME HISTCOPY,HIST 1
```

```
// RUN
// IFF  ?1?/ IF DATAF1-?1?  SCRATCH UNIT-F1,LABEL-?1?
// IFF  ?2?/ IF DATAF1-?2?  SCRATCH UNIT-F1,LABEL-?2?
// IFF  ?3?/ IF DATAF1-?3?  SCRATCH UNIT-F1,LABEL-?3?
// IFF  ?4?/ IF DATAF1-?4?  SCRATCH UNIT-F1,LABEL-?4?
// IFF  ?5?/ IF DATAF1-?5?  SCRATCH UNIT-F1,LABEL-?5?
// IFF  ?5?/ IF DATAF1-?6?  SCRATCH UNIT-F1,LABEL-?6?
// IFF  ?6?/ IF DATAF1-?7?  SCRATCH UNIT-F1,LABEL-?7?
// IFF  ?7?/ IF DATAF1-?8?  SCRATCH UNIT-F1,LABEL-?8?
// IFF  ?8?/ IF DATAF1-?8?  SCRATCH UNIT-F1,LABEL-?8?
// IFF  ?9?/ IF DATAF1-?9?  SCRATCH UNIT-F1,LABEL-?9?
// IFF  ?10?/  IF DATAF1-?10?  SCRATCH UNIT-F1,LABEL-?10?
// IFF  ?11?/  IF DATAF1-?11?  SCRATCH UNIT-F1,LABEL-?11?
// END
```

Saving History Files

by Thomas Straitwell and Martin Bell

Code on diskette:
Procedure HISTCOPY

When a S/36 is configured to save the history file periodically, the file is copied to a user file called HISTCOPY whenever the system history file becomes 80 percent filled. If the history file fills up twice in the same day, however, the system attempts to create a duplicate copy of the user file HISTCOPY and locks up the workstation until someone intervenes.

You can prevent this situation by using IBM's HISTCOPY procedure. If you add the OCL statements shown in Figure 8-28, the system creates history files named HIST.1, HIST.2, and so on, avoiding duplicate names. Procedure HISTCOPY uses an *n*-positional stack procedure to maintain *n* history files. As the system history file is saved, the *n*th file is deleted, and the others are renamed. We can change the size of the stack easily.

Figure 8-28
Procedure HISTCOPY

```
*
* HISTCOPY PROCEDURE  MAINTAINS AN N-POSITIONAL STACK OF HISTORY
*                     FILE ROLLOVERS
*                           P1  -  N POSITIONS COUNTER
*                           P2  -  N-1 COUNTER
*
* INITIALIZE COUNTERS
// EVALUATE P1,1-5 P2,1-4
* DELETE OLDEST FILE IN STACK
// IF DATAF1-HIST ?1? DELETE HIST ?1?,F1
* RENAME OTHER FILES IN STACK
// TAG LOOP
// IF DATAF1-HIST ?2? RENAME HIST ?2?,HIST ?1?
// EVALUATE P1,1-?1?-1 P2,1-?2?-1
// IF ?1?>1 GOTO LOOP
* RENAME NEWEST FILE
// RENAME HISTCOPY,HIST 1
```

Folders

CHAPTER

9

Maintaining Folders Automatically

by Ron Elliott

program by Matthew Henry

Code on diskette:
Procedure FOLDMK
RPG program FOLDMK
Screen format member FOLDMKFM
Message member FOLDMKMG

If you use S/36 folders to store documents, you know the value of organizing your documents; you're probably also familiar with the headaches that often accompany folder maintenance. Like user libraries, folders must be saved, restored, condensed, and reallocated. And you may have problems remembering folder names when trying to perform maintenance functions on all the system's folders.

One solution to the problem of keeping track of folder names is utility FOLDMK with prompt screen MAIN (Figure 9-1), which automatically tracks your system's folders and performs any SSP maintenance task — system service programs such as CONDENSE and SAVEFLDR — on every folder you've chosen. Utility FOLDMK consists of RPG program FOLDMK (Figure 9-2), screen format member FOLDMKFM (Figure 9-3), message member FOLDMKMG (Figure 9-4), and procedure FOLDMK (Figure 9-5).

Figure 9-1

Prompt screen MAIN

```
FOLDMK                              FOLDMK PROCEDURE

Type choices, press Enter

 Procedure to run                                     ________  SSP folder procedure

 Procedure parameters                                           Folder name assumed
           ____________________________________________          to be first parameter

 Name of file with "not to use" folder names          ________  Name of indexed file
 Name of procedure to create                          ________  Any valid procedure

 Name of library to place new procedure in            ________  Any existing library

 Delete procedure after it runs                       _         Y-Yes, delete it
                                                                N-No, keep it




Cmd3-Previous menu     Cmd7-Cancel procedure     Help-More information
```

Before Using Procedure FOLDMK

Before you can use utility FOLDMK, you must build an indexed file to contain the names of the folders you want to exclude from processing. Because IBM-supplied folders (i.e., #WPFLDR, #IWPCLD2, #IDDFLDR, #QRYFLDR, #WPDOCS, #OFCFLDR, #PROFLIB, #PRFFLDR, #IDDUSMP, and #USERDCT) generally remain static, routine maintenance on them is unnecessary, and you may want to exclude them from processing. You also may want to exclude specific user folders from automatic maintenance functions. Any label for the indexed file will do (our example's indexed file is NOTDFILE); however, the file must have a record length of eight bytes with the key defined over bytes 1 through 8.

Using Utility FOLDMK

Utility FOLDMK works by building a procedure to carry out the maintenance chores. A series of screens prompts you for information to build the procedure. The first screen, screen MAIN, prompts you for the name of the SSP procedure (e.g., SAVEFLDR or CONDENSE) that you want to be executed automatically for all folders. Screen MAIN also prompts for optional procedure parameters. Note that the SSP procedures check these parameters for errors when the procedures are executed. In the CONDENSE procedure, for example, the second (optional) parameter must contain the word FOLDER preceded by a comma.

The remaining prompts ask you for the name of the indexed file, the name of the procedure you are creating, the name of the library in which you want the newly created procedure stored, and whether you want to delete or save the procedure utility FOLDMK creates. You may want to save the procedure so you can execute it again without having to re-create it. Be aware, however, that the saved procedure contains only the names of folders that were on the system when utility FOLDMK created the procedure.

After responding to the prompts, press Enter for utility FOLDMK to create the procedure to perform the chosen SSP function on all nonexcluded folders. After utility FOLDMK finishes building the procedure, screen ASK2RUN (Figure 9-6) is displayed and asks whether you want to execute the procedure immediately at the workstation or execute it later via an EVOKE or the JOBQ.

Regular folder maintenance is a necessary burden, but utility FOLDMK frees you from the tedium of reading VTOC listings and helps you keep your folders slim and trim.

Figure 9-2

Program FOLDMK

```
*.  ..  1   . . 2  . . . 3  .   . 4   . . . 5     . 6   . .  7 .       8
H************************************************************************
H* Program. FOLDMK  Written by  Matthew P  Henry                        *
H* This program reads the VTOC directly and builds a procedure         *
H* for every folder on the system except ones found in the not-        *
H* to-do file                                                          *
H*---------------------------------------------------------------------*
H* Indicators:                                                         *
H* 10 - Used for NOTDFILE chain operation in subroutine SFLDR          *
H* 20 - Used for TESTB operation in SFLDR                              *
H*---------------------------------------------------------------------*
H* Switches.                                                           *
H************************************************************************
H      64                                                             FOLDMK
FPROCFILEO   F  80  80            DISK
FNOTDFILEIC  F   8   8R 8AI     1 DISK
E                    CNT     1   3 77   LNG     2 0
E                    OT           80  1
INOTDFILENS
IVTOCDS      DS
I                                       1   1 FFORG
I                                       3  10 FFLABL
I                                      11  160FFCRDT
I                                      17  17 FFTYPE
I                                      20  20 FFFLAG
I                                     119 126 FFPAR
I            DS
I                                       1  80 OTDS
I                                       1  80 OT
I           UDS
I                                       1   8 LPROC
I                                       9  54 LPARA
I                                      55  62 LNAME
I                                      63  70 LNEWL
I                                      71  71 LDELE
C* Initialization
C           LPROC     IFEQ 'CONDENSE'
C           LPARA     IFEQ *BLANKS
C                     MOVEL'FOLDER'  LPARA
C                     END
C                     END
C*
C                     MOVE *BLANKS   OTDS
C                     MOVELCNT,1     OTDS
C                     Z-ADDLNG,1     0
C                     MOVEALNAME     OT,0
C                     EXSR SFBLK
C                     MOVEACNT,2     OT,0
C                     EXCPTHEADER
C* Main control routine
C           FFORG     DOULE*BLANK
C                     MOVE *BLANKS   NAME    8
C                     EXSR SUBRVR
C           FFORG     IFGT *BLANK                      End of VTOC?
C                     EXSR SFLDR                       Select folder
C                     END                              End IFGT *BLANK
C                     END                              End DOULE *BLANK
C*
C           LDELE     IFEQ 'Y'                         Delete option?
C                     MOVE *BLANKS   OTDS              Blank output
C                     MOVELCNT,3     OTDS              move constant
C                     Z-ADDLNG,3     0                 set length
C                     MOVEALNAME     OT,0              move name
C                     EXSR SFBLK                       Find blank
C                     MOVEA',P,'     OT,0              Add P for REMOVE
C                     ADD  3         0                 set pointer
C                     MOVEALNEWL     OT,0              Move library name
C                     EXCPTDELOPT                      Output OCL
C                     END                              End IFEQ 'Y'
C* Termination
C                     EXCPTFOOTER
C                     SETON                     LR
C*-------------------------
C* Subroutines
C*
```

```
C* Select folder
C           SFLDR     BEGSR
C           FFLABL    CHAINNOTDFILE             10    Label in not do file
C   10      FFORG     IFEQ 'F'
C                     TESTB'1'       FFFLAG         20Must be folder DDA
C   20                DO
C                     MOVE *BLANKS   OTDS             Initialize work area
C                     MOVEL'// '     OTDS
C                     MOVEALPROC     OT,4             move procedure name
C                     MOVEAFFLABL    OT,13
C                     Z-ADD13        0       20
C                     EXSR SFBLK                      Find next blank
C           LPARA     IFNE *BLANKS                    Parameters empty?
C                     MOVEA','       OT,0             No  move comma
C                     ADD  1         0                    index up
C                     MOVEALPARA     OT,0                 move parameters
C                     END                             End IFNE *BLANKS
C                     EXCPTOFLDR                      Write proc line
C                     END                             End DO
C                     END                             End IFEQ 'F'
C                     ENDSR
C*
C* Call SUBRVR to open VTOC
C           SUBRVR    BEGSR
C                     EXIT SUBRVR                     Read VTOC
C                     RLABL          NAME             Name
C                     RLABL          VTOCDS           Return data
C                     ENDSR
C*
C* Find blank in work area OT
C* Setting 0 to start point speads up search
C           SFBLK     BEGSR
C                     MOVE '0'       $EOS    1        End of search flag
C           $EOS      DOUEQ'1'                        Search
C           OT,0      IFEQ *BLANK                     Blank?
C                     MOVE '1'       $EOS              Yes  end search
C                     ELSE                             No
C           0         IFEQ 80                          Pointer eq 80?
C                     MOVE '1'       $EOS              Yes  end search
C                     ELSE                             No
C                     ADD  1         0                   increment
C                     END                             End IFEQ 80
C                     END                             End IFEQ *BLANK
C                     END                             End DOUEQ'1'
C                     ENDSR
OPROCFILEE                HEADER
O                         OTDS      80
OPROCFILEE                HEADER
O                                   24 '// IF JOBQ-NO IF EVOKED-'
O                                   42 'NO EVALUATE P20-NO'
O* Folder output
OPROCFILEE                OFLDR
O                                   14 '// IF ?20?-NO '
O                                   37 '* ''Working with folder '
O                         FFLABL    45
O                                   46 ''''
OPROCFILEE                OFLDR
O                         OTDS      80
O* Footer to close procedure
OPROCFILEE                DELOPT
O                                   23 '// IF ?20?-NO * ''Removi'
O                                   47 'ng procedure from librar'
O                                   49 'y'''
OPROCFILEE                DELOPT
O                         OTDS      80
OPROCFILEE                FOOTER
O                                    7 '// CEND'
** Program constants                                                   Length
// COPY LIBRARY-P,NAME-                                                    24
,HIST-NO                                                                   09
// REMOVE                                                                  11
/*
```

Figure 9-3

Screen format member FOLDMKFM

```
*        1   -     2         3         4      .  5         6         7         8
0001 SMAIN                NY    63  Y                          CG123456
0009 DFL0001    79 1 2Y                        Y       CFOLDMK                 X
0010 D       FOLDMK PROCEDURE
0011 DFL0002    26 3 2Y                                CType choices, press EntX
0012 Der
0013 DFL0003    44 5 3Y                                CProcedure to run       X
0014 D
0015 DFL0004     8 54964 Y     51           51  Y N
0016 DFL0005    20 559Y                                CSSP folder procedure
0017 DFL0006    44 7 3Y                                CProcedure parameters   X
0018 D
0019 DFL0007    22 759Y                                CFolder name assummed
0020 DFL0008    44 81364 Y                      Y N                            X
0021 D
0022 DFL0009    21 860Y                                Cto be first parameter
0023 DFA0002    4410 3Y                                CName of file with "not X
0024 Dto use" folder names
0025 DFA0001     8104964 Y     53           53  Y N
0026 DFA0003    201059Y                                CName of indexed file
0027 DFA0001    4412 3Y                                CName of procedure to crX
0028 Deate
0029 DFA0002     8124964 Y     54           54  Y N
0030 DFA0005    191259Y                                CAny valid procedure
0031 DFA0003    4414 3Y                                CName of library to placX
0032 De new procedure in
0033 DFA0004     8144964 Y     55           55  Y N
0034 DFA0006    201459Y                                CAny existing library
0035 DFA0001    4416 3Y                                CDelete procedure after X
0036 Dit runs
0037 DFA0002     1164964 YA    56           56  Y N
0038 DFA0003    161659Y                                CY-Yes, delete it
0039 DFA0004    131759Y                                CN-No, keep it
0040 DFA0001    7023 2Y                                CCmd3-Previous menu     X
0041 DCmd7-Cancel procedure
0042 DFL0010    7024 264               Y               M
0043 SINFO      0100 Y             Y
0044 DFL0001    64 410Y                     Y          C                     INX
0045 DFORMATIONAL WINDOW
0046 DFL0002     1 510Y                     Y          C
0047 DFL0003    60 512Y                                C                       X
0048 D
0049 DFL0004     1 573Y                     Y          C
0050 DFL0005     1 610Y                     Y          C
0051 DFL0006    60 612Y                                M
0052 DFL0007     1 673Y                     Y          C
0053 DFL0008     1 710Y                     Y          C
0054 DFL0009    60 712Y                                C                       X
0055 D
0056 DFL0010     1 773Y                     Y          C
0057 DFL0011    64 810Y                 Y   Y          C                       X
0058 D                                Working
0059 SASK2RUN   0100      NY        Y                          CG123456
0060 DFL0001    53 613Y                     Y          C                   RUN X
0061 DPROCEDURE NOW
0062 DFL0002     1 713Y                     Y          C
0063 DFL0003    49 715Y                                C                       X
0064 D
0065 DFL0004     1 765Y                     Y          C
0066 DFA0003     1 813Y                     Y          C
0067 DFA0001    18 815Y                                CCreated procedure
0068 DFA0002     8 834Y  Y          Y
0069 DFA0004     9 843Y                                C Library
0070 DFA0002     8 853Y  Y          Y
0071 DFA0006     2 862Y                                C
0072 DFA0005     1 865Y                     Y          C
0073 DFA0006     1 913Y                     Y          C
0074 DFA0007    49 915Y                                C                       X
0075 D
0076 DFA0008     1 965Y                     Y          C
0077 DFL0005     11013Y                     Y          C
0078 DFL0006    221015Y                                CRun the procedure now?
0079 DFL0007     11038Y  YB    Y                Y N
0080 DFA0003    241040Y                                CY   - Run at terminal  X
0081 D
```

```
0082 DFL0008      11065Y                    Y          C
0083 DFL0009      11113Y                    Y          C
0084 DFA0005     491115Y                               C                      X
0085 D  N   - Do not run now
0086 DFL0011      11165Y                    Y          C
0087 DFA0007      11213Y                    Y          C
0088 DFA0001     491215Y                               C                      X
0089 D  0-5 - Place on job queue
0090 DFA0008      11265Y                    Y          C
0091 DFA0009      11313Y                    Y          C
0092 DFA0004     491315Y                               C                      X
0093 D  E   - Evoke procedure
0094 DFA0010      11365Y                    Y          C
0095 DFA0011      11413Y                    Y          C
0096 DFL0010     491415Y                               C                      X
0097 D
0098 DFA0012      11465Y                    Y          C
0099 DFL0012     531513Y                    Y          C  Enter-Continue     CmX
0100 Dd7-Cancel
```

Figure 9-4

Message member FOLDMKMG

```
FOLDMKMG.1
* Message for FOLDMK procedure and program
0001 ** Must have a procedure name, please enter a procedure name
0002 ** Must have a file name, please enter a file name
0003 ** File does not exist, reenter an existing name
0004 ** Must have a new procedure name, please enter a procedure name
0005 ** Must have a library name, please enter a library name
0007 ** Name is not a library, reenter a valid library name
0008 ** Delete prompt must be 'Y' or 'N', please reenter
* Text for INFO window
0101 Reading VTOC, please wait
0102 Copying procedure to specified library
* End of procedure messages
0201 Procedure has been evoked, press Enter
0202 Procedure has finished running, press Enter
0203 Procedure has been placed on the job queue, press Enter
```

Figure 9-5

Procedure FOLDMK

```
// MEMBER USER1-FOLDMKMG
*
* Screen procedure parameters can be changed by changing values below
*
// EVALUATE P3-#FOLDMK P4-?WS??TIME?  P5-#LIBRARY  P6-Y
*
// TAG BEGIN
// EVALUATE P64-1
// PROMPT MEMBER-FOLDMKFM,FORMAT-MAIN,LENGTH-'8,44,8,8,8,1,0,0,0,6'
// IFF ?CD?-0000 RETURN
// EVALUATE P51-  P52-  P53-  P54-  P55-  P56-  P63-1
// IF ?1?-                GOTO BEGIN ?1OF'0001U1'? ?51F'1'?
// IF ?3?-                GOTO BEGIN ?1OF'0002U1'? ?53F'1'?
// IFF DATAF1-?3?         GOTO BEGIN ?1OF'0003U1'? ?53F'1'?
// IF ?4?-                GOTO BEGIN ?1OF'0004U1'? ?54F'1'?
// IF ?5?-                GOTO BEGIN ?1OF'0005U1'? ?55F'1'?
// IFF DATAF1-?5?         GOTO BEGIN ?1OF'0006U1'? ?55F'1'?
// IFF LOAD-'#PTFLOG,?5?' GOTO BEGIN ?1OF'0007U1'? ?55F'1'?
// IFF ?6?-Y IFF ?6?-N    GOTO BEGIN ?1OF'0008U1'? ?56F'1'?
*
// TAG RUN
// EVALUATE P11-'0101U1'
// PROMPT MEMBER-FOLDMKFM,FORMAT-INFO,START-11,LENGTH-'6'
*
// LOCAL OFFSET-01,DATA-'?1?',BLANK-71
// LOCAL OFFSET-09,DATA-'?2?'
// LOCAL OFFSET-55,DATA-'?4?'
// LOCAL OFFSET-63,DATA-'?5?'
// LOCAL OFFSET-71,DATA-'?6?'
// EVALUATE P10-?WS??TIME?
// LOAD FOLDMK
// FILE NAME-NOTDFILE,LABEL-?3R?,DISP-SHRRM,IBLOCK-100,DBLOCK-100
```

```
// FILE NAME-PROCFILE,LABEL-?10R?,DISP-NEW,RECORDS-500,EXTEND-50
// RUN
*
// EVALUATE P11-'0102U1'
// PROMPT MEMBER-FOLDMKFM,FORMAT-INFO,START-11,LENGTH-'6'
// INFOMSG NO
// TOLIBR ?10?,F1,,,?5?,,,,ALL
// IF DATAF1-?10?  DELETE ?10?,F1
*
// INFOMSG YES
// EVALUATE P6=E
// PROMPT MEMBER-FOLDMKFM,FORMAT-ASK2RUN,START-4,LENGTH-'8,8,1'
// IFF ?CD?=0000  GOTO END
// IF      ?6?=E            EVOKE ?4?,?5?
// ELSE IF ?6?=Y            ?4?,?5?
// ELSE IF ?6?=0            JOBQ ?6?,?5?,?4?
// ELSE IF ?6?>0 IFF ?6?>5 JOBQ ?6?,?5?,?4?
// ELSE                     GOTO END
*
// TAG END
// RETURN
*******************************************************************
*       Name: FOLDMK
*    Purpose: Make procedure for working with all folders on a system
* Parameters: P1  - Name of procedure to run
*             P2  - Parameters for procedure
*             P3  - File name of "do not use" folder names
*             P4  - Name of new procedure created by FOLDMK
*             P5  - Library to place new procedure in
*             P6  - Y/N to delete new procedure after running it
*             P10 - Error message for prompt screen
*             ... - File name for FOLDMK procedure
*   Switches  NONE
*   LDA data: See program FOLDMK
```

Figure 9-6

Screen ASK2RUN

```
FOLDMK                          FOLDMK PROCEDURE

 Type choices, press Enter.
          *********************INFORMATIONAL WINDOW**********************
  Proced *                                                       * dure
          *  ##################RUN PROCEDURE NOW################         *
  Proced ***#                                                  #******** mmed
            # Created procedure  MKFOLD    Library: #LIBRARY   # rking   ameter
            #                                                  #
  Name of f # Run the procedure now? _ Y   - Run at terminal   # indexed file
            #                          N   - Do not run now    #
  Name of p #                          0-5 - Place on job queue # d procedure
            #                          E   - Evoke procedure   #
  Name of l #                                                  # ting library
            ##Enter-Continue####Cmd7-Cancel####################
  Delete procedure after it runs . .      . .   _          Y-Yes, delete it
                                                           N-No, keep it




 Cmd3-Previous menu     Cmd7-Cancel procedure     Help-More information
```

Reducing Folder Size

answered by Jeff Silden

Q We use DisplayWrite/36 (DW/36) for printing, spell checking, creating documents, and so on. I've noticed that the folders on which we use DW/36 develop a number of extents each day, hurting DW/36 performance. As a result, we use the ALLOCATE FOLDER (ALOC FLDR) procedure at the end of each day to eliminate the extents. However, sometimes the folder doesn't shrink. Why not? How can I reduce the folder to the smallest size possible?

A Your decision to use procedure ALOCFLDR is a good one because ALOCFLDR reorganizes the contents of the folder specified in parameter 1. It doesn't matter whether the folder contents are DW/36 documents, PS/36 mail logs, or IDDU definitions. However, make sure to specify MIN as the second parameter to procedure ALOCFLDR. If procedure ALOCFLDR's second parameter is left blank, the folder is organized, but its reorganization doesn't necessarily release free space contained within the folder. To reduce the folder to its minimum size, use the CONDENSE procedure before using ALOCFLDR, specifying FOLDER as the second parameter.

Also, you should allocate more space to the folder at the start of the day to minimize automatic extending. Then, periodically run the CONDENSE procedure against that folder to keep it as organized as possible.

IDDU and Query/36

CHAPTER

10

Printing an Enhanced Query Report Header Page

answered by George Applegate and Gary T. Kratzer

Code on diskette:
Procedure QQRYID
RPG program QQRYID

Q We have a problem with people running Query reports and then forgetting about them. Although everyone prints a cover page, it does not tell me who ran the reports. I've thought about setting up individual libraries for everyone who uses Query, but our space is limited. Is there a way to print the workstation or user ID on the cover page? Do you have any other suggestions?

A It would be nice if each user who runs a Query report would meet his or her responsibility to go pick it up. Barring that, Figures 10-1 and 10-2 contain a procedure and program that do the trick. Program QQRYID writes the user ID, workstation ID, date, and time to a printed header page. The printer statement CONTINUE-YES, which keeps the spool file open, enables the header page to be concatenated with the query report.

Figure 10-1
Procedure QQRYID

```
***************************************************************************
*   Create Query user printout for distribution purposes - User/WS/Date/Time   *
***************************************************************************
// LOCAL OFFSET-1,DATA-'?USER?
// LOCAL OFFSET-9,DATA-'?WS?'
// PRINTER CONTINUE-YES
// LOAD QQRYID
// RUN
***************************************************************************
QRYRUN VNAME,ELEVLIBR,VENDOR,PRINTER,........,RECSEL DETAIL
***************************************************************************
// PRINTER CONTINUE-NO
// LOCAL OFFSET-1,BLANK-10
```

Figure 10-2
Program QQRYID

```
•   1         2         3         4         5         6         7         8
H     064                      B                                     QQRYID
FPRINTER O   F 132 132            PRINTER
F*************************************************************
F*
F*  Write User ID and Workstation and date and time to a printed file
F*  Written by GEORGE APPLEGATE 2/90
F*
F*************************************************************
I           UDS
I                                        1   8 UDSUSR
I                                        9  10 UDSWS
C                     TIME           FLDDAT 120
C                     MOVELFLDDAT    UDSTIM  60
C                     Z-ADDFLDDAT    UDSDAT  60
C                     SETON                     LR
OPRINTER T  304   LR
O                                    6 'QQRYID'
O                                   75 'REPORT USER INFORMATION'
O        T  2     LR
O                                   10 'USER '
O                         UDSUSR    20
O                                   35 'WORKSTATION '
O                         UDSWS     38
```

```
0024 0                            50 'DATE '
0025 0                   UDSDATY  59
0026 0                            70 'TIME '
0027 0                   UDSTIM   79 '    .  '
```

Running Query/36 on the Job Queue

answered by Matthew Henry and Mark Rubinstein

Q Can I run Query from the job queue?

A To put a query on the job queue, simply take the // IF JOBQ statements out of the QRYRUN procedure. Or you can enter QRYRUN, in which case the screen will show four data lines. Fill in "Name of Query to Run" and "Name of Library Containing Query," and then press CMD4. Your query will be placed on the JOBQ. But one word of caution with this latter method: if after filling in the first two lines you press Enter instead of CMD4, you'll get additional questions — but the CMD4 will no longer work.

Deleting and Creating Files from Query/36

answered by Matthew Henry and Mark Rubinstein

Q How do I control file deletion/creation from Query?

A There are a couple of ways to control file deletion/creation from Query. If you specify the deletion/creation option as "create a new file" and then set autoresponse to answer QRY1032 with option 2, your users won't even get the message saying that the file already exists and they have the option of replacing it. If you are putting this all into a procedure, it would look something like this:

```
// RESPONSE MSG1032
// NOHALT 1,JOB
// QRYRUN parameters
// IF ?CD?>0000 code for file already existing
// RESPONSE QRY1032  (to restore message response values)
```

where source member MSG1032 contains:

```
QRY
1032 2,1
```

and source member QRY1032 contains:

```
QRY
1032 N
```

You also can use resource security to secure all your master files as update default access. This ensures that the file can't even be deleted by accident.

Converting Date Formats in Query/36, Part 1

by Mark Allen

In BitStop, September 1986, Robert Hughes presented a technique that allows S/38 Query users to convert dates from MMDDYY format to YYMMDD format. Mr. Hughes' technique will not work with Query/36 because Query/36 does not support a remainder function. However, the following three statements will accomplish the same conversion in Query/36 for field MDY in MMDDYY format:

```
REM  = MDY/100
REM1 = REM * 100
YMD  = REM + ((MDY - REM1) * 10000)
```

Field REM must be defined as a six-position field with no decimals, and field REM1 must be defined as an eight-position field with no decimals. The result field, YMD (also defined as six positions with no decimals), contains the date in YYMMDD format. As a specific example, let MDY equal 021086. REM is then equal to 0210 and REM1 equals 021000. The subtraction within the inner set of parentheses in the third statement yields 86, which is multiplied by 10,000 and added to REM, yielding 860210.

Converting Date Formats in Query/36, Part 2

by Rick Stanley

The computations shown below illustrate another approach Query/36 users can take for converting dates from MMDDYY format to YYMMDD format. Field DATE contains a date in MMDDYY format, and field YYMMDD contains the date in YYMMDD format. Fields Date1 and Date3 are defined as twelve-position fields; Date2 and YYMMDD are defined as six-position fields. All fields are defined with no decimal positions.

```
Date1  = DATE * 10000.01
Date2  = Date1/1000000
Date3  = Date2 * 1000000
YYMMDD = Date1 - Date3
```

For example, if DATE equals 021086, Date1 equals 210860210, Date 2 equals 210, Date3 equals 210000000, and finally YYMMDD equals 860210.

Creating RPG F-, I-, and O-Specs from IDDU with Query/36

answered by Matthew Henry

Q How can I access IDDU information to create file-, input-, and output-specifications for RPG II?

A Commercial packages are available, but IBM provides a way to extract F- and I-specs from IDDU in its Work with Data Files facility. You can edit any IDDU-defined file with the Query Data Entry facility (e.g., the QRYDE procedure or an option on the IDDUDISK "Work with Files" menu). When you invoke QRYDE, the prompt screen tells you a program is being built. Many people don't realize that when using QRYDE, you can copy the programs that have been built to another library for future editing. When QRYDE builds its necessary DFU programs, it also builds a library called #QDwsLBn, where *ws* is the QRYDE session's workstation-ID, and *n* is 1 for the main session or 2 for an inquiry session.

Library #QDwsLBn contains several items: the screen load module (#QDwsPGn), the DFU subroutine (#QDwsPGn), the DFU source specifications (#QDwsDFn), the DFU screen source (#QDwsPGn), and — perhaps most useful — RPG F- and I-specs (#QDwsPRn). You can copy all of these to a user library and then use the F- and I-specs in any RPG program.

To avoid later problems with QRYDE, make sure you do not delete the QRYDE library or any members in the library, and do not do anything with the library unless the QRYDE "Work with Data in a File" screen is displayed.

Creating RPG F- and I-Specs from IDDU

by Perry Gardai

program by William H. Dixon

Code on diskette:
Procedure IDDUXL
RPG program IDDUXL
Screen format member IDDUXLPM

Remaining true to your philosophies of implementing end-user computing and increasing programmer productivity, you have encouraged the development of independent Interactive Data Definition Utility (IDDU) and Query/36 applications throughout your organization. Obviously, this approach helps relieve the MIS backlog — until a user wants IDDU database information that is simply too complicated for Query/36.

Now you need an RPG application, and you are faced with the lack of standard RPG F-specs and I-specs to describe the files used in the Query/36 application. You could print the IDDU data dictionary and, from it, key in the RPG specifications. You could work your way through a series of cumbersome IBM-supplied routines to copy and modify the IDDU member (dodging error messages if your file uses more than 60 fields). Or, you could let utility IDDUXL do the translating.

Prompt screen IDDUXLPM (Figure 10-3) provides the user interface into the utility. Utility IDDUXL, which comprises prompt screen format member IDDUXLPM (Figure 10-4), procedure IDDUXL (Figure 10-5), and RPG program IDDUXL (Figure 10-6), converts IDDU members to

RPG F-specs and I-specs. Procedure IDDUXL controls the job flow, and program IDDUXL handles the actual translation process and creates the F and I source member file.

Figure 10-3

Prompt screen IDDUXLPM

```
                         IDDUXL PROCEDURE

               Translate IDDU specs into RPG F & I specs

Enter name of data dictionary . . . . . . . . . . . . . . . . . .  #USERDCT

Individual format name (blank for all)  . . . . . . . . . . . . .

Enter name of member to create  . . . . . . . . . . . . . . . . .  IDDUFI

Enter name of library to contain member . . . . . . . . . . . . .  TEST

Include file/format/field descriptions  . . . . . . . . . . . Y,N  N




Cmd3-Previous menu     Cmd4-Put on job queue     Cmd9-Evoke
```

Getting Started

To run the utility, you simply key in IDDUXL at a command line, and prompt screen IDDUXLPM appears. The prompt screen supplies procedure IDDUXL with the variable information it needs to perform the IDDU-to-RPG translation according to your requirements. The variable parameters are as follows:

Parameter 1 — Name of user dictionary containing the IDDU specifications.

Parameter 2 — Name of specific member to be converted to RPG specifications. This parameter is left blank if all members are to be converted.

Parameter 3 — Name of the new source member that will contain the converted specifications.

Parameter 4 — Name of the library to contain the new source member.

Parameter 5 — File, format, field descriptive information — include Y or N.

The prompt screen contains predetermined defaults for parameters 1, 3, 4, and 5, which you should change to suit your preferences.

Proceeding

Once you provide the variable information, procedure IDDUXL tests for requests for end of job (i.e., Command key 3 and Command key 7), checks each parameter for validity, and tests to see whether the remainder of the procedure should be placed on the job queue or evoked. If procedure IDDUXL finds no end-of-job requests or parameter errors, it calls IBM procedure IDDUPRT to output the selected data dictionary to a spooled printer file named IDDUPRT, which is automatically put on hold because of the PRIORITY-0 parameter. The $UASF utility then copies this spool file into data file ?WS?.IDDU.

In preparation for the execution of program IDDUXL, procedure IDDUXL stores the name of the source member that will contain the converted specifications (parameter 3) into the LDA and tests parameter 5 to determine whether descriptive information is to be included, setting Switch 1 accordingly. Program IDDUXL then reads the ?WS?.IDDU data file and outputs RPG F- and I-specs into data file ?WS?.TMP1.

Reshuffling

Program IDDUXL probably looks more complicated than it actually is. Keep in mind that the function of program IDDUXL is to read each record from the IDDU specifications, determine its function, and output the appropriate F- and I-specs into a source member data file. The apparent complexity of the program results from the fact that the IDDU specifications in data file ?WS?.IDDU are not in the proper order for the final production of RPG specifications.

To keep track of the specifications' relative position, procedure IDDUXL creates work file ?WS?.WRK, and program IDDUXL outputs a record to ?WS?.WRK whose key and data fields indicate the relative position that input records from ?WS?.IDDU should occupy in the RPG F and I source member. Once determined, program IDDUXL writes this information into ?WS?.TMP1 output file records.

Procedure IDDUXL uses output file ?WS?.TMP1 as input to a COPYDATA routine that organizes the file into the proper sequence, resulting in file ?WS?.TMP2. The COPYDATA routine uses parameter 5 from the prompt screen to determine the record length of the source member to be produced. If parameter 5 is Y, meaning the descriptive information is to be included in the source member, the routine creates ?WS?.TMP2 as a 120-character file. Otherwise, a 96-character file is created.

The final step in the process is the execution of a TOLIBR procedure. This routine uses file ?WS?.TMP2 to create the RPG specifications source member — with the name specified by parameter 3 of the prompt screen — in the library specified by parameter 4.

If you're curious to see how program IDDU does its job, print a copy of IDDUPRT, temporarily change the work and sort files to permanent files, run the program, and compare the two files against the IDDUPRT printout.

With utility IDDUXL, you can effectively use IDDU databases in your mainstream RPG applications. All you need is an editor like SEU, DSU, or FSEDIT to include the new, IDDUXL-created F-specs and I-specs in the RPG programs your users require.

Figure 10-4

Screen format member IDDUXLPM

```
*       1 ..   . 2 . . .  3 .       4     .. 5 ..    . 6  .      7     .. 8
SPROMPT01           06YY                                     CDGI
D          16 133Y                                   CIDDUXL PROCEDURE
D          41 318Y                                   CTranslate IDDU specs inX
Dto RPG F & I specs
D          63 6 2Y                                   CEnter name of data dictX
Dionary       .                 .
D           8 667Y  Y                     Y
D          63 8 2Y                                   CIndividual format name X
D(blank for all)     .          . .
D           8 867Y  Y                     Y
D          6310 2Y                                   CEnter name of member toX
D create     . . .              .   .   .
D          81067Y  Y                      Y
D          6312 2Y                                   CEnter name of library tX
Do contain member     . . .   .
D          81267Y  Y                      Y
D          6314 2Y                                   CInclude file/format/fieX
Dld descriptions     . .   .       .   Y,N
D          11467Y  YA                     Y
D          1823 2Y                                   CCmd3-Previous menu
D          212325Y                                   CCmd4-Put on job queue
D          102351Y                                   CCmd9-Evoke
DERRMSG    5024 2Y                Y
```

Figure 10-5

Procedure IDDUXL

```
** PROCID IDDUXL - TRANSLATE IDDU SPECS INTO RPG F & I SPECS
**
**----------------------------------
*
// IF JOBQ-YES     GOTO $RUN
// IF EVOKED-YES   GOTO $RUN
*
// EVALUATE ?1'#USERDCT'?  ?3'IDDUFI'?  ?4'?SLIB?'?  ?5'N'?
// EVALUATE P6-
*
// TAG $PMT
// PROMPT MEMBER-IDDUXLPM,FORMAT-PROMPT01,LENGTH-'8,8,8,8,1,50'
// IF ?CD?-2003  RETURN
// IF ?CD?-2007  CANCEL
*
// TAG $NOPMT
// EVALUATE P6-
*
// IF '?6?'-   IF  ?1?-                                  +
               EVALUATE P6-'Invalid data dictionary name - must not be blank  '
// IF '?6?'-   IFF DATAF1-?1?                            +
               EVALUATE P6-'Label entered does not exist                      '
// IF '?6?'-   IF  DATAF1-?1?  IF  LOAD-'#PTFLOG,?1?'  +
               EVALUATE P6-'Label entered is a library                        '
// IF '?6?'-   IF  ?3?-                                  +
               EVALUATE P6-'Member name must be entered                       '
// IF '?6?'-   IF  ?4?-                                  +
               EVALUATE P6-'Library name must be entered                      '
// IF '?6?'-   IFF DATAF1-?4?                            +
               EVALUATE P6-'Label entered does not exist                      '
// IF '?6?'-   IF  DATAF1-?4?  IFF LOAD-'#PTFLOG,?4?'  +
               EVALUATE P6-'Label entered is not a library                    '
// IF '?6?'-   IF  SOURCE-'?3?,?4?'                      +
               EVALUATE P6-'Source member already exists in specified library '
// IF '?6?'-   IFF ?5?-Y       IFF ?5?-N                 +
               EVALUATE P6-'Invalid response - must be Y or N                 '
*
// IFF '?6?'-  GOTO $PMT
*
// IF ?CD?-2004 JOBQ 1,,IDDUXL,?1?,?2?,?3?,?4?,?5?
// IF ?CD?-2004 RETURN
// IF ?CD?-2009 EVOKE IDDUXL *ALL.
// IF ?CD?-2009 RETURN
*
// * 'IDDUXL procedure is running'
// INFOMSG NO
*
```

```
// TAG $RUN
*
// PRINTER PRIORITY-0,FORMSNO-?WS?XX,NAME-IDDUPRT
IDDUPRT ?1'#USERDCT'?,ALL,FILE,?2'ALL'?
*
// LOAD $UASF
// RUN
// SPOOL SPOOLID-F?WS?XX,NAME-?WS?.IDDU,RELCANS-CANCEL,RETAIN-J
// END
*
// LOCAL OFFSET-1,DATA-'?3?',BLANK-8
*
// IF ?5?-Y  SWITCH 10000000
//    ELSE   SWITCH 00000000
*
// LOAD IDDUXL
// FILE NAME-INPUT,LABEL-?WS?.IDDU,RETAIN-J
// FILE NAME-OUTPUT,LABEL-?WS?.TMP1,RECORDS-100,EXTEND-100,RETAIN-J
// FILE NAME-WRKFILE,LABEL-?WS?.WRK,BLOCKS-100,EXTEND-100,RETAIN-J
// RUN
*
// IF ?5?-Y  EVALUATE P5-120
//    ELSE   EVALUATE P5-96
COPYDATA ?WS?.TMP1,,?WS?.TMP2,,,,J,REORG,,,,,?5?,,S
*
TOLIBR ?WS?.TMP2,F1,,,?4?
```

Figure 10-6

Program IDDUXL

```
*. .    1 . . . . 2 . .  . 3 . . . . 4 . .  . . 5 . . . . . 6 . . .  . . . 7 . .  . . 8
0001 H/TITLE IDDUXL - TRANSLATE IDDU SPECS INTO RPG F & I SPECS
0002 H      64                        B       1                               IDDUXL
0003 F*
0004 F* ----- INDICATOR LIST -----
0005 F*  U1 - Include file/format/field descriptions if on
0006 F*  01 - Line: "Short comment---..." for files
0007 F*  02 - Line  "Short comment---..." for formats
0008 F*  03 - Line· Containing field information
0009 F*  04 - Line. "No record ID codes have been defined for this format."
0010 F*  05 - Line: Containing record ID codes
0011 F*  90 - General purpose indicator
0012 F*  91 - On if disk file; Off if communications file
0013 F*  92 - On if first line of record ID code
0014 F*
0015 FINPUT   IP  F 150 150            DISK
0016 FOUTPUT  O   F 150 150 17AI    134 DISK                      U
0017 FWRKFILE UF  F  16  16 12AI      1 DISK                      A
0018 I*
0019 IINPUT   NS       12 CD  13 Ce  14 Cf
0020 I       AND       15 Ci  16 Cn  37 C-
0021 I       AND       38 C
0022 I                                       39  46 FNAME
0023 I*
0024 I        NS       12 CF  13 Ci  14 Cl
0025 I                                       39  45 FTYPE
0026 I*
0027 I       NS       12 CM  13 Ca  14 Cx
0028 I                                       39  42 FLENX
0029 I*
0030 I        NS  01   12 CS  13 Ch  14 Co
0031 I       AND       15 Cr  16 Ct  38NC-
0032 I                                       39  82 FDESC
0033 I*
0034 I        NS  02   12 CS  13 Ch  14 Co
0035 I       AND       15 Cr  16 Ct  38 C-
0036 I                                       42  85 RDESC
0037 I*
0038 I        NS  03   40 C   41 CC  42 CH
0039 I       AND       43 CA  44 CR  45 C
0040 I       OR        40 C   41 CZ  42 CO
0041 I       AND       43 CN  44 CE  45 C
0042 I       OR        40 C   41 CP  42 CA
0043 I       AND       43 CC  44 CK  45 C
0044 I       OR        40 C   41 CB  42 CI
```

```
I       AND      43 CN  44 C   45 C
I                                      14  19 VNAME
I                                      25  28OVBEG
I                                      33  36OVLEN
I                                      38  38 VDEC
I                                      41  44 VTYPE
I                                      49  92 VDESC
I*
I        NS  04  20 CN  21 Co  22 C
I       AND      30 CI  31 CD  32 C
I        NS  05  36 CE  37 CQ  38 C
I       OR       36 CN  37 CE  38 C
I       OR       36 CZ  37 C   38 C
I       OR       36 CN  37 CZ  38 C
I       OR       36 CD  37 C   38 C
I       OR       36 CN  37 CD  38 C
I                                      12  14 ANDOR
I                                      18  23 VNAME
I                                      29  32 IDPOSX
I                                      36  37 IDTEST
I                                      45  45 IDVAL
I        NS
I*
IWRKFILE NS
I                                      13  16OVBEG
I*
I            DS
I                                       1   4 ALPHA
I                                       1   10ALPHN1
I                                       2   4 ALPHT1
I                                       1   20ALPHN2
I                                       3   4 ALPHT2
I                                       1   30ALPHN3
I                                       4   4 ALPHT3
I                                       1   40ALPHN4
I*
I            DS
I                                       1  12 KEY
I                                       1   40FNUM
I                                       5   60RNUM
I                                       7  12 VNAME
I*
I           UDS
I                                       1   8 MEMBER
C*
C   01                EXSR $FILE
C   02                EXSR $FORMT
C   03                EXSR $FIELD
C   05                EXSR $IDCOD
C*
C*---------------------------------
C*
C           $FILE     BEGSR
C*
C*  This subroutine performs the logic for the file records
C*
C*  Set on indicator 91 if this is a disk file (as opposed to a
C*  communications file)  Only output records if it is a disk file
C*
C           FTYPE     COMP 'DISK                    91
C*
C*  Convert the file length from a left-justified alpha field to
C*  a numeric field
C*
C                     MOVE FLENX     ALPHA
C                     EXSR $ATON
C                     Z-ADDNUMBER    FLEN    40
C*
C*  Increment the file number, and initialize the record number
C*
C                     ADD  1         FNUM
C                     Z-ADDO         RNUM
C*
C           #FILE     ENDSR
C*
```

```
C*---------------------------------
C*
C          $FORMT    BEGSR
C*
C*  This subroutine performs the logic for the format records
C*  NOTE - The record-type identification records (I-specs) are not
C*  written to the output file until the record selection records
C*  are processed
C*
C*  Increment the format number, and initialize the format record
C*  selection line number
C*
C                    ADD  1         RNUM
C                    Z-ADD0         RNUMX   20
C*
C          #FORMT    ENDSR
C*
C*---------------------------------
C*
C          $FIELD    BEGSR
C*
C*  This subroutine performs the logic for the field records
C*
C*  Determine variable type to be placed in column 43 of the input
C*  specs (P, B or blank)   If field is packed or binary, determine
C*  actual number of characters taken in the record
C*
C                    MOVE ' '       VTYP    1
C          VTYPE     IFEQ 'PACK'
C                    MOVE 'P'       VTYP
C                    DIV  2         VLEN
C                    ADD  1         VLEN
C                    END
C*
C          VTYPE     IFEQ 'BIN '
C                    MOVE 'B'       VTYP
C          VLEN      IFLE 4
C                    Z-ADD2         VLEN
C                    ELSE
C                    Z-ADD4         VLEN
C                    END
C                    END
C*
C*  Determine the ending position
C*
C                    Z-ADDVBEG      VEND    40
C                    ADD  VLEN      VEND
C                    SUB  1         VEND
C*
C*  Increment the field number
C*
C                    ADD  1         VNUM    40
C*
C*  Add a record to the work file containing the field and its
C*  beginning position
C*
C          KEY       CHAINWRKFILE              90
C   90               EXCPT#WRKA
C  N90               EXCPT#WRKU
C*
C          #FIELD    ENDSR
C*
C*---------------------------------
C*
C          $IDCOD    BEGSR
C*
C*  This subroutine performs the ID code logic
C*
C*  Set on indicator 92 if this is a continuation line
C*
C          ANDOR     COMP *BLANKS                  92
C*
C*  Convert the ID position from a left-justified alpha field to
C*  a numeric field
C*
```

```
C                     MOVE IDPOSX    ALPHA
C                     EXSR $ATON
C                     Z-ADDNUMBER    IDPOS   40
C*
C*  Get the field's beginning position from the work file, and
C*  determine the actual position of the test character.
C*
C           KEY       CHAINWRKFILE              90
C   90                Z-ADDO         VBEG
C                     ADD  VBEG      IDPOS
C                     SUB  1         IDPOS
C*
C*  Determine the test codes to place on the input spec based on
C*  the codes in the IDDU printout (NZ & ND do not need conversion)
C*
C           IDTEST    IFEQ 'EQ'
C                     MOVE ' C'      IDTEST
C                     END
C*
C           IDTEST    IFEQ 'NE'
C                     MOVE 'NC'      IDTEST
C                     END
C*
C           IDTEST    IFEQ 'Z '
C                     MOVE ' Z'      IDTEST
C                     END
C*
C           IDTEST    IFEQ 'D '
C                     MOVE ' D'      IDTEST
C                     END
C*
C*  Increment the format record selection line number.
C*
C                     ADD  1         RNUMX
C*
C           #IDCOD    ENDSR
C*
C*---------------------------------
C*
C           $ATON     BEGSR
C*
C*  This subroutine converts a left-justified alpha field into a
C*  numeric field
C*
C           ALPHT1    IFEQ *BLANKS
C                     Z-ADDALPHN1    NUMBER  40
C                     ELSE
C           ALPHT2    IFEQ *BLANKS
C                     Z-ADDALPHN2    NUMBER
C                     ELSE
C           ALPHT3    IFEQ *BLANK
C                     Z-ADDALPHN3    NUMBER
C                     ELSE
C                     Z-ADDALPHN4    NUMBER
C                     END
C                     END
C                     END
C*
C           #ATON     ENDSR
C*
C*---------------------------------
C*
OOUTPUT  D        1P
O                                  23 '// COPY LIBRARY-S,NAME-'
O                        MEMBER    31
O                                 150 '0 0000 00 00 0000'
OOUTPUT  D        91 01
O                                   6 'F'
O                        FNAME     14
O                                  16 'IP'
O                                  19 'F'
O                        FLEN  Z   23
O                        FLEN  Z   27
O                        FTYPE     46
O                    U1  FDESC    118
```

```
0                                       134 '1'
0                            FNUM       139
0                                       142 '00'
0                                       145 '00'
0                                       150 '0000'
OOUTPUT  D         91 03
0                                         6 'I'
0                            VTYP        43
0                            VBEG   Z    47
0                            VEND   Z    51
0                            VDEC        52
0                            VNAME       58
0                     U1     VDESC      118
0                                       134 '2'
0                            FNUM       139
0                            RNUM       142
0                                       145 '99'
0                            VNUM       150
OOUTPUT  D         91 04
0                                         6 'I'
0                            FNAME       14
0                                        16 'NS'
0                            RNUM        20
0                     U1     RDESC      118
0                                       134 '2'
0                            FNUM       139
0                            RNUM       142
0                                       145 '01'
0                                       150 '0000'
OOUTPUT  D         91 05
0                                         6 'I'
0                     92     FNAME       14
0                     92                 16 'NS'
0                     92     RNUM        20
0                    N92     ANDOR       16
0                            IDPOS Z     24
0                            IDTEST      26
0                            IDVAL       27
0                     U1     RDESC      118
0                                       134 '2'
0                            FNUM       139
0                            RNUM       142
0                            RNUMX      145
0                                       150 '0000'
0*
OOUTPUT  T         LR
0                                         7 '// CEND'
0                                       150 '9 9999 99 99 9999'
0*
OWRKFILE EADD                #WRKA
0                            KEY         12
0                            VBEG        16
0*
OWRKFILE E                   #WRKU
0                            VBEG        16
```

Creating IDDU File Definitions

by Matthew Henry

Creating IDDU file definitions using IBM's IDDU utility can be difficult and awkward. I have found an easy way to create IDDU file definitions, although at first my process may seem backwards. Start with IDDUDFN, and select file definitions. Then select the option to create a new file definition. On the "Define a File Definition" screen, choose the option to select formats for a file. When the "Select and Sequence Formats for a File" screen appears, enter the name of the as yet nonexistent format definition in the format definition name field. IDDU then displays a prompt indicating it cannot find the definition and asking you to press Enter to create it.

Pressing Enter displays the format definition screen; pressing Enter on this screen then brings up the "Select and Sequence Fields for a Format" screen. With SSP Release 5.1, you can enter the key word ALL to create multiple field definitions. After you have created all the field definitions, press Enter enough times to go back through the format and file definition screens. The IDDU file description is created automatically, and the format definition is added to the file definition automatically.

Defining S/36 Filler Fields

by Sarah E. McBride

The help text under filler fields in section 4.1.3.2.4. of IDDU help text enlightened me on the subject of defining filler fields for my IDDU record formats. Filler fields are reserved or ignored space that is not shown to the user if the format definition is used during a query, remains in the field record's buffer, and has no name. You can use fillers to account for unused space in a record (i.e., space reserved for future expansion) or to hide certain fields from Query/36 for security purposes.

A filler field is represented on the SELECT AND SEQUENCE FIELDS FOR A FORMAT display by an asterisk in the NAME column of the field definition list (Figure 10-7). Use the Field definition name and the Sequence number prompts to add a filler field. Type a sequence number that indicates where you want to place the filler field. Specify *nnnnn for the field definition name, where *nnnnn* is a number between 1 and 4095 that indicates the number of positions to reserve. Then press ENTER. To remove a filler field, use the SEQ column. Type blanks over its sequence number field, and press ENTER.

Figure 10-7

Select and sequence fields for a format display

```
SEQ   NAME    BEGIN   LENGTH
10    FIELD1     1       7
20    *          8       5     <--- Filler field at position
30    FIELD2    13      10           8 of format, 5 positions
long
```

Updating IDDU Definitions

by Jeff Silden

Q When I try to save changes to an IDDU definition for a file that is already linked, I get a halt with the message:

```
'IDDU-0299: Definition cannot be saved'
```

If I go into inquiry and try to unlink the file, I get the message:

```
'IDDU-0402: Dictionary currently in use'
```

even though *I'm* the user. How can I save the IDDU work when I discover, too late, that the file is linked?

A The error messages you mention prevent users from making changes that might affect another user's data. The next time you are unable to save IDDU revisions, try the procedure in the next paragraph while observing the following cautions. Although the technique below is quite reliable, you must be quick about it. Nobody else can be in the dictionary or the file while you are using the technique. Should you discover, however, that the dictionary is damaged because someone else "sneaked in" a dictionary change while you were using the technique, simply copy the dictionary's file definitions using the IDDU functions themselves.

Now for the technique. Close the IDDU session by pressing the Attention key, and select option 2 to cancel the job and close the files. Immediately unlink the offending file, or files, and go right back into the IDDU session, directly to the field, format, or file definition you are trying to save. IDDU prompts you with a "Recover Interrupted Session" panel from which you should choose option 1 (Recover Session). After that, you are returned to the panel just before the save panel, and you now can ask IDDU to save your dictionary work.

Libraries

CHAPTER

11

Retrieving a Library's Users

by Perry Gardai

program by Matthew Henry

Code on diskette:
Procedure TESTUL
RPG program TESTUL
Assembler subroutine SUBRUL

The S/36 utility TESTUL determines which workstations or jobs are using a particular library and gives you a tool upon which to build new utilities. The TESTUL utility uses an assembly language subroutine to present you with a scrolling screen that shows individual library use.

As a S/36 programmer, you know how frustrating it is to try to perform a CONDENSE, or a library RENAME, or a library DELETE, only to be stymied by a system message such as "SYS-2582 *library name* — This library not condensed, being used." Unfortunately, IBM provides no easy way to determine who is using a particular library — especially if the library is tied up by evoked jobs or jobs on the job queue. Although the IBM-supplied D U (display user status) operator command displays all the users of all your libraries, this information is disordered and can overwhelm you with detail. Thus, we present the S/36 TESTUL utility, which does provide an easy and effective method of determining library use from any terminal on the system. More important, TESTUL returns information about just the library you specify.

The TESTUL utility consists of procedure TESTUL (Figure 11-1) and program TESTUL (Figure 11-2), which calls subroutine SUBRUL. For ease of access, procedure TESTUL and program TESTUL should be stored in #LIBRARY.

Using the TESTUL Utility

To use the TESTUL utility, simply key in

```
TESTUL libname
```

where *libname* (parameter 1) is the name of the library to be checked for current users. Procedure TESTUL loads the library name into the LDA, beginning in position 247. The TESTUL utility uses LDA positions 201 through 257 to avoid conflict with the LDA positions POP uses. Procedure TESTUL initializes to zero parameter 2, which serves as a loop counter, and loads it into the LDA starting in position 255. Then procedure TESTUL calls program TESTUL, a one-cycle RPG program that calls SUBRUL via the EXIT operation and three RLABL statements. The first RLABL statement contains the library name you specified. Subroutine SUBRUL retrieves information about one user of this library and stores the user information in data structure JOBDS, named in the third RLABL statement. (This data structure must be at least 46 bytes long to hold all the information SUBRUL returns. If the data structure is not long enough, SUBRUL will not return any data.)

Because the specified library could have several users, SUBRUL allows repetitive calls to retrieve information about each of them. The second RLABL statement, JOB#, specifies the user for which SUBRUL should return information. Field JOB# contains 0 to return information about the first job using the specified library, 1 for the second job, 2 for the third job, and so on. After calling SUBRUL, program TESTUL copies the contents of the JOBDS data structure (information about a user of the library) into LDA positions 201 through 246 via the LJINFO field, and procedure TESTUL displays this user information on your workstation screen. Then procedure TESTUL increments the counter, parameter 2, and repeats the process until position 209 of the LDA (corresponding to field JOBNAM in data structure JOBDS) is blank. This loop is repeated as often as jobs are found running from the specified library and results in a scrolling screen of messages that display all users of the specified library.

When position 209 of the LDA is blank (i.e., no other jobs are using the specified library), the procedure performs a final test of parameter 2. If parameter 2 is 0 at this time, no workstation or job is using the specified library, and a message is issued accordingly. (If parameter 2 is a value other than 0, no additional message is issued.) Procedure TESTUL then terminates.

As with any user members stored in an IBM-supplied library (e.g., #RPGLIB or #LIBRARY), you should remember that subroutine SUBRUL, program TESTUL, and procedure TESTUL will be removed from the system each time you install a new release of SSP. Therefore, you should keep a copy of all the components of this utility in your toolkit library so you can readily replace them after you install a new release.

The TESTUL utility is an example of tool building using a core tool as a building block to create a new tool. The core tool, SUBRUL, could be implemented as a standalone assembler program, but it is implemented as a subroutine to incorporate into other tools. You can use this tool-building technique to build completely new tools or to use one tool in different ways. For instance, you could incorporate the TESTUL utility directly into the IBM-supplied CONDENSE procedure to show a list of jobs using a library *before* you get the SYS-2582 message.

With the TESTUL utility, you can avoid the hassles of trying to figure out who is using the library that you want to use. And you also can use these concepts of tool building to enhance your programming efforts.

Figure 11-1

Procedure TESTUL

```
* Find out who's using a library
// INFOMSG YES
// LOCAL OFFSET-247,DATA-'?1R?',BLANK-8
// EVALUATE P2,3-0
// * 'The following jobs are using library ?1? '
// TAG LOOP
// LOCAL OFFSET-255,DATA-'?2?'
// LOAD TESTUL
// RUN
// IF ?L'209,1'?-  GOTO DONE
```

```
// * 'Job ?L'209,8'?, user ?L'201,8'?, Proc ?L'217,8'?, Running ?L'233,8'??L'225,8'?'
// EVALUATE P2 ?-?2?+1
// GOTO LOOP
// TAG DONE
// IF ?2?=000 * '(nobody)'
// PAUSE
```

Figure 11-2

Program TESTUL

```
*         1         2         3         4         5         6         7         8
00001H     064                       8    1                                  TESTUL
00002 *
00003 *- This program tests SUBRUL by retrieving job information for a job
00004 *    using a specified library
00005 *
00006I            JDS
00007I                                               201 246 LJINFO
00008I                                               247 254 LIBNAM
00009I                                               255 257OJOB#
00010IJOBDS        DS
00011I                                                 1   8 USERID
00012I                                                 9  16 JOBNAM
00013I                                                17  24 FSTPRC
00014I                                                25  32 CURPRC
00015I                                                33  40 PRGNAM
00016I                                                41  46OJSTIME
00017I                                                 1  46 JINFO
00017C                    SETON                         LR
00018C                    EXIT SUBRUL
00019C                    RLABL          LIBNAM
00020C                    RLABL          JOB#
00021C                    RLABL          JOBDS
00022C                    MOVE JINFO     LJINFO
```

Re-creating Subroutine SUBRUL

If you don't have assembler subroutine SUBRUL, you can re-create it with procedure MKSUBRUL (you don't need IBM's Assembler Language Program Product to install SUBRUL). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRUL. You need to run MKSUBRUL only once because SUBRUL is subsequently linked into program TESTUL when it is compiled.

```
// * 'Re-creating R-module SUBRUL in library @RPGLIB '
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'     Number of #MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9E4D34040000000400000000000060400000000900032500000000087'
// LOCAL OFFSET-273,DATA-+
'08072103910000009038930000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-@RPGLIB
// END
* Patch the new SUBRUL member to insert object code
// LOAD #FEPTX
// RUN
HDR  3800 SUBRU00000
PTF  1CAF @SUBRUL,09,,@RPGLIB
DATA 6F88 00 0000 E2E4C2D9E4D3403000001F900F8000000000000000000000000000000000000000
DATA E82C 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 1307 00 0040 E33003293408020233401020CA340202C2EF8808035010202750202C0702EA0075
DATA 092C 00 0080 03082C0202800C400010A02E2C08202B6790203C34A2000000202B27211913080700
```

Continued

```
DATA 4801 00 0080 E333025D02F0BC4025AC242425F4000F8A700036A102D5F2812875A2B736A202
DATA 5287 00 00A0 D5F2811B8D070902EAF2010A070202ED02DEC082026CB5A200312D2B241C1201
DATA 7924 00 00C0 E33702951CF1B71F75A111F1B72FC08702BE35A202F09C07070C9C0717749C07
DATA AF13 00 00E0 1F7C9C0727B49C0109698C050F02DB98020A6A98030B6A98020C6B98002A110D
DATA 00C1 00 0100 E33702CD030D6B98020E6C98030F6C8C052D02D898022865980329659B022A66
DATA 8860 00 0120 9B032B6698022C6798032D670E0102D202E0F68000C2A10000C2A200002D280F
DATA A380 00 0140 E31402E200C087000000000000F0F0F0F0F0F0F0F1000D002D0000000000000000
DATA 4E78 00 0160 0000000000000000000000000000000000000000000000000000000000000000
DATA 588A 00 0180 C5FFFFE200000000000000000000000000000000000000000000000000000000
DATA 59B3 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 32D8 00 01C0 615C000000000000000000000000000000000000000000000000000000000000
DATA 54A3 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
END  05FA
```

Testing for Library Existence

by Tom McLendon

Have you ever wished for a conditional procedure expression to allow you to verify the existence of a library? Sure, you can use // IF DATAF1-libname, which will be true if a file by the specified name is on disk. But // IF DATAF1-libname doesn't verify that what is found is a library.

I have found a simple way to check for the existence of a library. Because all libraries have a "hidden" load member — #PTFLOG (used to record any PTFs applied to the library) — you can use the OCL statement // IF LOAD-#PTFLOG,libname... to check for the existence of that member.

Retrieving Library Directory Information

by Gary T. Kratzer

Code on diskette:
RPG code SMPLD
Assembler subroutine SUBRLD

Assembler subroutine SUBRLD lets you read library directory entries from within an RPG program, which eliminates the need to run $MAINT and output directory entry data to a file every time you need to retrieve a directory entry. Although subroutine SUBRLD does not represent a major breakthrough in the type of library and directory information you can retrieve, it does provide an easier and more flexible way to retrieve the information you need — when you need it. You can use subroutine SUB-RLD to retrieve library member information such as the member's attributes, the number of statements in a member, or the number of sectors the member occupies. In addition, you can use subroutine SUBRLD to retrieve information about an entire library, which could be helpful for tasks such as reallocating the size of a library.

Using subroutine SUBRLD in an RPG program requires you to code an EXIT SUBRLD operation, which is followed by five required RLABL statements (Figure 11-3) that constitute the subroutine's parameters.

Depending on the parameters you use, you can use SUBRLD to retrieve directory information in a variety of ways. Subroutine SUBRLD's parameters are as follows:

• LIBNAM (library name) — an eight-byte field that contains the library name (left justified) of the library in which the desired member resides.

• MEMNAM (member name) — an eight-byte field that can contain the name (left justified) of the desired library member. In addition, MEMNAM controls how the search of the directory entries takes place. For example, if MEMNAM contains a member name, the directory information for that member is returned. If MEMNAM contains a partial name (a partial name is followed by an asterisk — e.g., SUBR*), the next directory entry that matches the partial name for the specified member type is returned. If MEMNAM contains *LIBR, SUBRLD retrieves general information about the entire library. (Note that the fields in the DIRDS (Directory Data Structure) (described below) are different when you request information for an entire library.) If MEMNAM is blank, the next directory entry is read for the specified member type.

• MEMTYP (member type) — a one-byte field that contains the member type of the desired library members. Specify O for object, P for procedure, R for subroutine, or S for source.

• DIRDS — a data structure that contains either the directory entry or library information returned by subroutine SUBRLD. Depending on the type of information requested, DIRDS must be at least 70 to 80 bytes long. The DIRDS format for obtaining specific directory entries is shown in Figure 11-4a; the DIRDS format for obtaining information about the entire library is shown in Figure 11-4b. The fields contained within both the directory entry data structure and the library information data structure are listed in Figure 11-5.

The fields in these two data structures that return hexadecimal values (e.g., LBBLIB (first sector of library) or LBELIB (last sector of library)) are actually character fields. All attribute bytes are binary data except for attribute-byte five (subtype), which is returned in hex representation. You can use the TESTB (Test the Bit) instruction to find which bits are set in each attribute byte.

Additional information about the fields in a library directory, as well as information about an entire library, can be found in *System Data Areas* (LY21-0592).

• RCODE (return code) — a one-byte field that contains the return codes, which are 0 for normal return, 1 for library not found, 2 for member not found or end of members (for partial/sequential searches), and 3 for data structure too small. If you read directory entries sequentially or perform partial searches, SUBRLD returns a 2 in the RCODE field upon reaching the end of the library members list. You can repeat the search by simply calling SUBRLD again.

Figure 11-6 contains a program that first sequentially reads all the directory entries in library TESTLIB and then retrieves information for the library. If you think about it, there are probably many jobs that could be made easier by subroutine SUBRLD's ability to retrieve detailed information from within an RPG program. So the next time you need library or directory information, think subroutine SUBRLD.

Figure 11-3

Calling sequence for subroutine SUBRLD

```
*   . 1         2         3         4         5   .     6         7         8
C                         EXIT SUBRLD
C                         RLABL           LIBNAM  8          Input
C                         RLABL           MEMNAM  8          Input
C                         RLABL           MEMTYP  8          Input
C                         RLABL           DIRDS  80          Output
C                         RLABL           RCODE   1          Ouput
```

Figure 11-4a

Format of DIRDS for a library member

```
*.      1         2         3         4         5         6         7         8
IDIRDS          DS
I                                              1   1 DRTYPE
I                                              2   9 DRNAME
I                                             10  15 DRADDR
I                                             16  18 DR#TXT
I                                             19  22 DRLINK
I                                             23  27 DR#STM
I                                             28  31 DRSCA
I                                             32  33 DRRLD
I                                             34  36 DRCORE
I                                             37  37 DRATR1
I                                             38  38 DRATR2
I                                             39  39 DRATR3
I                                             40  41 DRMRT
I                                             42  43 DRREL
I                                             44  46 DRTOTL
I                                             47  47 DRATR4
I                                             48  53 DRMOD
I                                             54  59 DRDATE
I                                             60  63 DRTIME
I                                             64  65 DRATR5
I                                             66  69 DRPTF@
I                                             70  70 DRATR6
```

Figure 11-4b

Format of DIRDS for an entire library

```
*       1         2         3         4         5         6         7         8
IDIRDS          DS
I                                              1   6 LBFMT1
I                                              7  11 LBLBSZ
I                                             12  15 LBDRSZ
I                                             16  21 LBUSEC
I                                             22  27 LBASEC
I                                             28  32 LBUDIR
I                                             33  37 LBADIR
I                                             38  43 LBBLIB
I                                             44  49 LBELIB
I                                             50  55 LBBDIR
I                                             56  61 LBEDIR
I                                             62  67 LBBMEM
I                                             68  73 LBEMEM
I                                             74  79 LBNMEM
I                                             80  80 LBEXTN
```

Figure 11-5

The fields within the directory data entry structure and the library information data structure

DRTYPE	Member type. (O, P, R, S)
DRNAME	Member name.
DRADDR	Disk address of member. (Hex)
DR#TXT	Number of text sectors. (types O, R) Record length. (types P, S)
DRLINK	Link edit address. (Hex)
DR#STM	Number of statements in member. (types P, S)
DRSCA	Start control address, entry point. (Hex)
DRRLD	RLD displacement. (Hex)
DRCORE	Core required, in sectors.
DRATR1	Attribute byte 1. (Binary)
DRATR2	Attribute byte 2. (Binary)
DRATR3	Attribute byte 3. (Binary)
DRMRT	MRTMAX count. (type O) If MRT proc, contains hex FF.
DRREL	Release level.
DRTOTL	Total number of sectors in module.
DRATR4	Attribute byte 4. (Binary)
DRMOD	Reference number.
DRDATE	Date member was changed/created. (YYMMDD)
DRTIME	Time member was changed/created. (HHMM)
DRATR5	Member subtype. (Hex)
DRPTF@	Displacement of PTF table in member. (Hex)
DRATR6	Attribute byte 6. (Binary)

DIRDS for an entire library:

LBFMT1	Format-1 address. (Hex)
LBLBSZ	Library size in blocks.
LBDRSZ	Directory size in sectors.
LBUSEC	Used member sectors.
LBASEC	Available member sectors.
LBUDIR	Used directory entries.
LBADIR	Available directory entries.
LBBLIB	First sector of library. (Hex)
LBELIB	Last sector of library. (Hex)
LBBDIR	First sector of directory. (Hex)
LBEDIR	Last sector of directory. (Hex)
LBBMEM	First sector of members. (Hex)
LBEMEM	Last sector of members. (Hex)
LBNMEM	Next available member sector. (Hex)
LBEXTN	Contains a Y if library extent exists.

Figure 11-6

Sample code that reads all library directory entries. (This code appears on diskette as member SMPLD.)

```
*        1         2         3         4         5         6         7         8
I*
I*- Data structure for individual member gets
I*
IDIRDS       DS
I                                            1   1 DRTYPE
I                                            2   9 DRNAME
I                                           10  15 DRADDR
I                                           16  18 DR#TXT
I                                           19  22 DRLINK
I                                           23  27 DR#STM
I                                           28  31 DRSCA
I                                           32  33 DRRLD
I                                           34  36 DRCORE
I                                           37  37 DRATR1
I                                           38  38 DRATR2
I                                           39  39 DRATR3
I                                           40  41 DRMRT
I                                           42  43 DRREL
I                                           44  46 DRTOTL
I                                           47  47 DRATR4
I                                           48  53 DRMOD
I                                           54  59 DRDATE
I                                           60  63 DRTIME
I                                           64  65 DRATR5
I                                           66  69 DRPTF@
I                                           70  70 DRATR6
I*
I*- Data structure for entire library gets
I*
ILIBDS       DS
I                                            1   6 LBFMT1
I                                            7  11 LBLBSZ
I                                           12  15 LBDRSZ
I                                           16  21 LBUSEC
I                                           22  27 LBASEC
I                                           28  32 LBUDIR
I                                           33  37 LBADIR
I                                           38  43 LBBLIB
I                                           44  49 LBELIB
I                                           50  55 LBBDIR
I                                           56  61 LBEDIR
I                                           62  67 LBBMEM
I                                           68  73 LBEMEM
I                                           74  79 LBNMEM
I                                           80  80 LBEXTN
C*
C*- Read sequentially through all member types
C*
C                     MOVE 'TESTLIB 'LIBNAM  8         Set library name
C                     MOVE *BLANKS   MEMNAM  8         Blank name - sequential search
C*
C                     MOVE '0'       MEMTYP  1         Object
C                     EXSR GETDIR
C*
C                     MOVE 'P'       MEMTYP            Proc
C                     EXSR GETDIR
C*
C                     MOVE 'R'       MEMTYP            Subroutine
C                     EXSR GETDIR
C*
C                     MOVE 'S'       MEMTYP            Source
C                     EXSR GETDIR
C*
C*- Now get info for entire library
C*
C                     MOVE '*LIBR   'MEMNAM            Library request
C                     EXIT SUBRLD
C                     RLABL          LIBNAM
C                     RLABL          MEMNAM
C                     RLABL          MEMTYP
```

```
C                   RLABL          LIBDS
C                   RLABL          RCODE
C*
C*- Subroutine to read members until a "2" is returned in RCODE
C*
C         GETDIR    BEGSR
C         RCODE     DOUEO'2'
C                   EXIT SUBRLO
C                   RLABL          LIBNAM
C                   RLABL          MEMNAM
C                   RLABL          MEMTYP
C                   RLABL          DIRDS
C                   RLABL          RCODE
C*
C*        (At this point DIRDS contains the directory info
C*         for the next member  Insert code as needed )
C*
C                   END
C                   ENDSR
```

Re-creating Subroutine SUBRLD

If you don't have assembler subroutine SUBRLD, you can re-create it with procedure MKSUBRLD (you don't need IBM's Assembler Language Program Product to install SUBRLD). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRLD. You need to run MKSUBRLD only once to create the SUBRLD subroutine.

```
// * 'Re-creating R-module SUBRLD in library #RPGLIB
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000295'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9D3C44040000001 20DD00000000060400000000990009200000000 1388'
// LOCAL OFFSET-273,DATA-+
'0814125531000000803D330000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-A,TO-#RPGLIB
// END
* Patch the new SUBRLO member to insert object code
// LOAD $FEFIX
// RUN
HDR  388C SUBRLO0000
PTF  9279 RSUBRLO,98,,#RPGLIB
DATA 0E47 00 0000 E208E2E4C2D803C400000018FE00000000000000000000000000000000000000
DATA 980A 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 6828 00 0040 E3320032F2870FE2E4C20903C440F148F24040404040340104793402047034 08
DATA 17A2 00 0060 04813DE8188CF201203C0618BC380706423807105C38070000302C2821101916
DATA 327F 00 0080 E332006505163807028 0C2A21050F4010401F40152023CF018AA350104817501
DATA 06FD 00 00A0 021D07188200F281381C07188200C2A204FA9C070700AF00202820191509 0601
DATA 1F21 00 00C0 E3310097020A0AF401041980020A186AF2010E0F0718821 8B23CF118AAC08704
DATA F950 00 D0E0 603C00188 33C00082E2C02083C0A3501048175010540020282420 1C1814120B
DATA 611A 00 0100 E32F00C707001888C00101F2350104814001 0D1873F202083CF318AAC0870460
DATA F88F 00 0120 76010F35A205040F021899188 92C0018893 50E0200002C2826221817100 80703
DATA 6C06 00 0140 E33100F918 99188F3C0318A8C08704824C05061 88F7C404F8B101CF290037CE8
DATA 335F 00 0160 4F2C0218992CC08704CE4C040A1 8A62C021899262F02002E2A2520100807 0301
DATA 5245 00 0180 E330012A1899290E02189918 6DC08704CE4C030E18A734A11892C2A105078C02
DATA 4368 00 01A0 1528F4014075A20F35A118922C0118891C2C00189700302827191511 0C080601
```

Continued

```
DATA E748 00 01C0 E32E015939C08704CE4C05141BA72C011898212C0018973AC08704CE4C061A18
DATA 2349 00 01E0 A70F021899189B2C0118990EC08704CE4C041F0000002B2522201C17120D0904
DATA 84E6 00 0200 E32E01881BA70F02189918892C01188910C08704CE4C04241BA72C021889033C
DATA B61E 00 0220 0318A8C08704B24C062A189F2C021899063C0300002B27221E18151008070501
DATA AFF8 00 0240 E32F01881BA8C08704B24C06301B9F2C021899093C031BA8C08704B24C05381B
DATA 3C70 00 0260 9F2C0218990C3C031BA8C08704B24C063C1B9F2C002E2926201C17130E0A0501
DATA AFCC 00 0280 E32F01E8021899173C031BA8C08704B24C05421B9F2C021B991A3C031BA8C087
DATA 8A39 00 02A0 04B24C054B1B9F2C021B991F3C031BA8C08704B2002F2B26221D1914100B0702
DATA 1808 00 02C0 E332021B4C054E1B9FC087048010071B8B00F281531C071B8B00C2A11B843C00
DATA 2194 00 02E0 1B83C2A2062E8C0000AF07080BBC000BE202013C0018B60032211D19140C0804
DATA 8C0E 00 0300 E332024E4D07071B83F281067D5C00F201080C0006361B98F2871A9C00000002
DATA 6E42 00 0320 0101E202013FFF1896700000F101280F071B881B883501000000302E24131104
DATA C56B 00 0340 E33B028B04B1750108130001BB300F2812A1C001BB300C2A2082E8C0008C0209
DATA F1CE 00 0360 7DD600F201038C08097D0900F201036C04097D0700F20103BC01090016100B01
DATA 4564 00 0380 E33202B8C2A2062EC0871068B8800BF2100C3C0018B33CF218AAC0870480B6A2
DATA 982E 00 03A0 11C2A100003501048140010018 71F2020C3C0018B33CF30030282419151107 03
DATA 57A9 00 03C0 E32F02E81BAAC08704807B010F8C0000008C070B082C02189908 3C031BA8C087
DATA 5A5C 00 03E0 04B24C050E189F0F02189918892C0018990CC0870000002C2826221D19140601
DATA 8880 00 0400 E330031804CE4C021118A74403161878 30D61883F201122C0118980E3C0218A8
DATA 5587 00 0420 C08704B24C0316189044041A18783DE21883F28107002029241F18160F080601
DATA E10C 00 0440 E32E03483DD71883F201140F02189918892C0118880EC08704CE4C041A18A72C
DATA E12A 00 0460 011898103C0218A8C08704B24C031E1B9D2C000000002C27231E1A16100C0A03
DATA 2098 00 0480 E32F03781887113C0118A8C08704B24C0120189844112318 7830D81883F20114
DATA 4C82 00 04A0 0F02189918892C00189912C08704CE4C022318A70D2F2A25211F18140F0A0801
DATA 5D88 00 04C0 E93203A98C0024138C0025148C002615440128187830D618B3F201140F021899
DATA 0391 00 04E0 18892C00188918C08704CE4C012818A730D718B3F20118002F2826211D181410
DATA F257 00 0500 E33103DD80FF18F201122C001B99163C0118A8C08704B24C0128189E44012A18
DATA D801 00 0520 786802281768032A170F02189918892C01189919C08700002E2A281C17120E09
DATA C67E 00 0540 E338041404CE4C022D18A78C002E1A440634187868022F1C6803301C68023110
DATA 249F 00 0560 6803321D6802331E6803341E44063A18786802351F6803361F6802002C0F0B01
DATA 1970 00 0580 E336044A372068033820680238216803 3A2144033E1878680238226803 3C2268
DATA 1571 00 05A0 0230236B033E232C001897243C0118A8C08704B24C014018982C00342F282812
DATA 188C 00 05C0 E392047D011B98263C0218A8C08704B24C0344189D6C004627C2A10000360104
DATA CA87 00 05E0 817B01124C00001BAA0E01048118 6FC2A10000C2A20000002A28241C10080702
DATA C786 00 0600 E33404B2C0870000340B04CD34A1189234A21895C2A11B9AC2A218974C010118
DATA ED18 00 0620 78680200006803 01007DF900F204037F39007DF901F204037F001C17130F0807
DATA F78A 00 0640 E33004E33801D20102E202013F0118A8F1012C35A1189235A21895C087000034
DATA E3F7 00 0660 0804F9040718A718783C1818A9080718A718A70E0200002E2C2824221E161208
DATA 3507 00 0680 E31604F918981899F220043A0118A73F0118A9F1011AC0870000000000000000
DATA 3508 00 06A0 000000000000000000000000000000000000000000000000000000000E0A0301
DATA E838 00 06C0 E300050540000000000000000000000000000000000000000000000000000000
DATA 0914 00 06E0 0000000000000000000000000000000000000000000000000000000000000000
DATA 2805 00 0700 E31F0526C9C60000000000040000A100A080052E000000000000000000000000
DATA 813E 00 0720 000000000000000000000000000000000000000000000000000000000000000F
DATA 4941 00 0740 E300063810000000000000000000000000000000000000000000000000000000
DATA F120 00 0760 0000000000000000000000000000000000000000000000000000000000000000
DATA 7796 00 0780 E30306438006600A000000000000000000000000000000000000000000000000
DATA 777E 00 07A0 0000000000000000000000000000000000000000000000000000000000000002
DATA 8748 00 07C0 E31010800858D4C1D3C8D54040C0801068800000000000000000000000000000
DATA A573 00 07E0 000000000000000000000000000000000000000000000000000000000000000C
DATA 8625 00 0800 E32718BF00000000000100130045004FF0F0F0F0F0F0F04040404040404040
DATA 711F 00 0820 5CD3C9C209404040E8001A0F0000000000000000000000000000000000000000
DATA 7CC0 00 0840 E33A18F600E2E4C2D9D3C4404040C39897A898888788A34040835040F1F8F8F9
DATA 66E0 00 0860 6840C78199A840E34840D29981A3A885984040C19383409989B7B8A3A2409900
DATA 8E4D 00 0880 E30618FDB5A2B599A5B584000000000000000000000000000000000000000000
DATA 8F31 00 08A0 0000000000000000000000000000000000000000000000000000000000000000
DATA E548 00 08C0 C50000F000000000000000000000000000000000000000000000000000000000
DATA 052C 00 08E0 0000000000000000000000000000000000000000000000000000000000000000
END 8683
```

Listing Members Created or Modified Within Given Date Range

by Perry Gardai

program by James Harr

Code on diskette:
Procedure PRGLST
RPG program PRGLST

I'm sure you all know how easy it is to plunge into a project for a month or so, and then when the day of reckoning comes (when a status report to management is due), you scratch your head and wonder, "Now exactly what have I done for the past month?" While numerous evaluation forms and various other tools have been developed to aid in this type of assessment, most are very difficult to administer. The S/36 PRGLST utility may be just what you are looking for to document the ongoing progress of DP development efforts within your organization.

The PRGLST utility is a simple yet effective tool for listing the names of all library members that have been added to or changed within a specific library during a specified period of time. The listing (Figure 11-7) shows the member name, the type, the date and time a member was created or changed, and the number of changes. The utility produces the listing by manipulating a library directory listing produced by the $MAINT utility. The PRGLST utility prompts for the name of the library to be listed and then for the range of dates on which to report. After the operational parameters have been established, a $MAINT routine produces a library directory listing that is put on hold within the spooler automatically. Then, the $UASF utility transforms the directory listing into a disk file. Once in disk file format, the directory listing is sorted by library member name, within date, within library type sequence. Finally, an RPG program takes the sorted version of the disk file that contains the library directory and produces the report as seen in Figure 11-7.

To use the PRGLST utility, key the procedure and program into either #LIBRARY or into your programmer's tool box library. Then key in PRGLST, and you are on your way.

Rather than using a prompt screen, procedure PRGLST (Figure 11-8) uses screen message statements and required substitutional parameters to establish all operational variables within the procedure. The first screen message asks the user to supply parameter 1 (?1R?), the name of the target library to be listed. The next two statements validate that the response is indeed a library. If it is not, a message to that effect is issued, and the procedure is reset.

After the target library has been established, the procedure establishes the date range for the report. The date is established in three parts, parameters 2, 3, and 4. Parameter 2 (?2R?) is the year of the desired time period, and if it is not supplied by the user, it defaults to 90. Parameter 3 (?3R?) is the month beginning the desired period; it defaults to January (i.e., 01) if the user does not supply an alternate value. Finally, parameter 4 (?4R?) is the month ending the period for the report. It defaults to 12, December, if not otherwise specified.

At this point, all the operational parameters have been established, but before the real work can begin, the two work files that will be developed within the procedure, SPROG and PPROG, are deleted from disk if they should already exist. Next, the procedure calls the $MAINT utility, which outputs a copy of the target library's directory to file PPROG.

File PPROG is then sorted into the proper predetermined sequence and includes only those library members that occur within the specified date ranges. Although the five $GSORT Include statements within the sort specifications may seem a bit complex at first, they are actually very straightforward. The first two check to see that two slash (//) marks appear in each desired record. The slash marks are embedded with the member creation date in each data record and differentiate these records from other records within the file, such as the header and trailer records. The next three Include statements verify whether each record falls within the desired time period.

The sort field specifications sequence the output file as follows. Position 12 of each record contains the member type (e.g., O = Object, P = Procedure) and is the primary sequence field. The next three field specifications sequence the members in YYMMDD date order (field positions 22 through 29). Field positions 1 through 8 contain the member name itself, which is the final sequence field. The final field specification designates that the output file SPROG is to contain the first 55 characters of the input file.

Once sorted, file SPROG is passed to print program PRGLST (Figure 11-9). The program is very straightforward. Indicator L1 designates a control break on field TYPE. Using this indicator in the O-specs causes the listing to double-space between library types as they are printed. As each record of file SPROG is read, one line is printed. Each print line contains the library member name, the library type, the last date and time a member was changed, and the number of changes to date.

The utility has one limitation with the user-specified year in parameter 4. Because only one year is specified, the utility will not span year-end boundaries. Therefore, to produce a listing of program changes made from December 1985 through February 1986, you would have to run the utility twice. Each run would have to specify the appropriate year for the report.

You also should be aware that the number of changes reported represents changes to date, not the number of changes since the last time the report was run against a specific library. To report the number of changes since the last time the procedure was run, you would have to reset the change counter to zero manually after the report is run. Conceptually, you could consider any library member with a change level of zero to be production ready. Any member with a change counter greater than zero, therefore, would represent the number of changes since the program was put into a production environment.

For those of you responsible for tracking the progress of development efforts within your DP department, procedure PRGLST represents an effective and efficient tool for management reporting, programmer productivity analysis, and project control.

Figure 11-7

Sample listing of new and changed programs by utility PRGLST

```
USEQTY     O     01/03/86     15.44     15
IV0102     O     01/16/86     09.37     49
RPGOBJ     O     01/21/86     13.33     17
AR0106     O     01/22/86     14.04      4
IVB020     O     02/03/86     10.35      2

USQTY      P     01/06/86     15.23     60
RS0005     P     01/28/86     16.04      4
SORTCUST   P     01/31/86     11.46      9

REP101     R     01/10/86     13.20      1
REP134     R     01/10/86     13.50      1
BLANKPRE   R     01/16/86     10.90      2
CHECK1     R     01/24/86     16.30      1

IV0102     S     01/16/86     08.59     49
OE0054     S     01/21/86     13.25     17
AR0106     S     01/22/86     14.02      4
IVB020     S     02/03/86     10.29      2
```

Figure 11-8

Procedure PRGLST

```
// * 'Library Name'
// IFF LOAD-'#PTFLOG,?1R?' PAUSE '?1? Is not a Library'
// IFF LOAD-'#PTFLOG,?1R?' RESET PRGLST
// * 'Year , defaults to 90'
// IF ?2R?- EVALUATE P2-'90'
// * 'Starting Month , defaults to 01'
// IF ?3R?- EVALUATE P3-'01'
// * 'Ending Month    , defaults to 12'
// IF ?4R?- EVALUATE P4-'12'
// IF DATAF1-SPROG DELETE PPROG,F1
// IF DATAF1-PPROG DELETE PPROG,F1
*
// LOAD $MAINT
// FILE NAME-INPUT,LABEL-PPROG,UNIT-F1,RETAIN-J
// RUN
// COPY TO-PRINT,FROM-?1?,PRTFILE-PPROG,NAME-ALL,LIBRARY-ALL,DISPLAY-DIRINFO
// END
*
// LOAD #GSORT
// FILE NAME-INPUT,LABEL-PPROG,RETAIN-S
// FILE NAME-OUTPUT,LABEL-SPROG,UNIT-F1,RECORDS-?F'A,PPROG'?
// RUN
    HSORTR    15A        3X  45
    I C  24  24EQC/
    IAC  27  27EQC/
    IAC  28  29EQC?2?
    IAC  22  23GEC?3?
    IAC  22  23LEC?4?
    FNC  12  12
    FNC  28  29
    FNC  22  23
    FNC  25  26
    FNC   1   8
    FDC   1  45
// END
*
// LOCAL BLANK-*ALL
// LOCAL OFFSET-1,DATA-'?1?'
// LOAD PRGLST
// FILE NAME-SPROG,DBLOCK-50,RETAIN-S
// RUN
```

Figure 11-9

Program PRGLST

```
*... ... 1 ... . 2 .. .. 3 .:. . 4 .. . 5 . ... 6 . 7 . 8
0001 H      008                     B                                   PRGLST
0002 FSPROG    IP  F  45  45           DISK
0003 FPRINTER O    F 132 132     OF    PRINTER
0004 ISPROG    NS  01
0005 I                                        1   8 PROG
0006 I                                       12  12 TYPE  L1
0007 I                                       22  29 DATE
0008 I                                       32  36 TIME
0009 I                                       39  440CHANGS
0010 ILDA         UDS
0011 I                                        1   8 LIBR
0012 OPRINTER H  101   1P
0013 O        OR       OF
0014 O                                 6 'PRGLST'
0015 O                                60 'LISTING OF CHANGED PROGR'
0016 O                                67 'AMS FOR'
0017 O                       LIBR     76
0018 O                       UDATE Y  90
0019 O                               125 'PAGE'
0020 O                       PAGE  Z 132
0021 O          H  2       1P
0022 O        OR           OF
0023 O                                15 'PROGRAM'
0024 O                                20 'TYPE'
0025 O                                30 'DATE'
0026 O                                40 'TIME'
0027 O                                50 'CHANGES'
0028 O          D  1       01
0029 O                       PROG     15
0030 O                       TYPE     20
0031 O                       DATE     30
0032 O                       TIME     40
0033 O                       CHANGSZ  50
0034 O          T  1       L1
0035 O                               132 ' '
```

Retrieving Source and Procedure Members from a Library

by Gary T. Kratzer

Code on diskette:
RPG code SMPSG
Assembler subroutine SUBRSG

Back in the days of the S/3, IBM provided an assembler subroutine to read source and procedure members directly from a library into RPG programs. This subroutine came bundled with the RPG compiler, just as SUBR21, SUBR22, and others are provided today with the S/36. But when the S/34 was introduced, the subroutine for reading library members, for whatever reason, disappeared. Since that time, we have not had any method for reading source and procedure members directly from a library into RPG programs.

Most programmers must read source and procedure members from a library at one time or another. Normally, you do this by using $MAINT to copy the member to a disk file and to read this file into an RPG program. Unfortunately, a few drawbacks to this method exist. One, the program must "look out" for the // COPY and // CEND delimiter statements that $MAINT places in the file. Two, if more than one member needs to be read, you usually need a separate disk file for each one. And three, even if the entire member does not need to be read, $MAINT still must spend the time to copy every source statement into the disk file.

The somewhat crude $MAINT method of reading library members does get the job done — but slowly and clumsily. Fortunately, there is a more efficient, convenient, and flexible way to read source and procedure members: use the Source Get feature of the library maintenance routines built into the SSP. But because access to these routines is not provided by high-level languages, you need an assembler language interface to perform the task. Such an interface is subroutine SUBRSG, which lets an RPG program read from any library any source or procedure member, even multiple library members simultaneously.

To use subroutine SUBRSG in an RPG program, you must code an EXIT SUBRSG operation, which must be followed by either four or six RLABL statements, depending on the type of call you are making. The two types of calls — Open request and Get Next request — and their respective parameters are described below.

The Open Request

Before subroutine SUBRSG can read a library member, the member must be opened with an Open request. An Open operation first checks for the existence of the specified library and whether the member is in that library. If the member is found, it is opened, and subroutine SUBRSG then can retrieve the member's text records with Get Next requests. The format of an Open request is shown in Figure 11-10, and the parameter descriptions are as follows:

• OP — one-byte field that contains the operation to be performed. For an Open request, OP should contain the letter O.

• LIBNAM — eight-byte field that contains the library name where the member resides.

• MEMNAM — eight-byte field that contains the name of the member to open.

• MEMTYP — one-byte field that contains the type of library member to open. Specify S for a source member or P for procedure.

• PLIST — 39-byte field that, upon returning from subroutine SUBRSG, contains the parameter list that corresponds to the member you just opened. This parameter list is used as input to subroutine SUBRSG on all subsequent Get Next calls. You should never alter the contents of this field. Your only responsibility is to keep track of which parameter list goes with which library member. Also, because PLIST contains mostly binary data, you should not attempt to display it.

• RCODE — one-byte field that contains the return code. The return codes for an Open request are:

0 — Open request successful. Okay to issue Get Next requests.

1 — Library not found.

2 — Member not found or corrupted member.

The Get Next Request

Once the member has been opened successfully, you may issue Get Next requests to read its text records from the library. The format of a Get Next is shown in Figure 11-11, and the parameters are as follows:

• OP — one-byte field that contains the letter N for Get Next. If OP does not contain O, N is assumed.

• PLIST — 39-byte field that contains the parameter list corresponding to the member you want to read. You should save the contents of PLIST after every call to subroutine SUBRSG.

• TEXT — 120-byte field that contains the next text record, left justified, from the requested member.

• RCODE — one-byte field that contains the return code. The return codes for a Get Next request are:

0 — Successful Get operation.

3 — End of member or corrupted member.

Subroutine SUBRSG reads sequentially through members until you reach the end or you want to stop. There is no close operation. If at any time while reading a member you want to start over with the first record, simply issue another Open request before continuing to issue Get Next requests.

You can use subroutine SUBRSG to access an unlimited number of library members simultaneously within the same program. To access several members at one time, save the contents of PLIST after opening each member. Then, when you want to read records from a particular member, supply the parameter list that corresponds to that member when you call subroutine SUBRSG on a Get Next request. Don't forget to save the contents of PLIST after every call to subroutine SUBRSG because the parameter list changes after each record is retrieved. An example of code that reads multiple members is shown in Figure 11-12.

Figure 11-10

Calling sequence for SUBRSG for an open request

```
*    1         2         3         4         5         6         7         8
C                        EXIT SUBRSG
C                        RLABL          OP      1         Input
C                        RLABL          LIBNAM  8         Input
C                        RLABL          MEMNAM  8         Input
C                        RLABL          MEMTYP  1         Input
C                        RLABL          PLIST  39         Output
C                        RLABL          RCODE   1         Output
```

Figure 11-11

Calling sequence for SUBRSG for a get next request

```
*    1         2         3         4         5         6         7         8
C                        EXIT SUBRSG
C                        RLABL          OP      1         Input
C                        RLABL          PLIST  39         Input/Output
C                        RLABL          TEXT  120         Output
C                        RLABL          RCODE   1         Output
```

Figure 11-12

Sample code that reads multiple members. (This code appears on diskette as member SMPSG.)

```
*    1         2         3         4         5         6         7         8
E*
E                       PLST        5 39               Save area for 5 parm lists
E                       PROC       10120               Proc save array
E                       SRC        10120               Source save array
E*
C*
C                        Z-ADD1         Z                 Init index
C                        MOVE '0'       OP                Open request
C*
C*- First open proc CUS001 in ARLIB
C*
C                        MOVE 'ARLIB   'LIBNAM            Library name
C                        MOVE 'P'       MODTYP            Type - Proc
C                        MOVE 'CUS001  'MODNAM            Proc name
C                        MOVE *BLANKS   PLIST             Clear parm list
C                        EXSR OPEN                        Open the member
C*
C              RCODE     IFEQ '0'                         If open successful
C                        MOVE PLIST     PLST,Z             Save parm list
C                        ADD  1         Z                  Bump index
C                        END
C*
C*- Then open source CUS001 in ARLIB
C*
C                        MOVE 'S'       MODTYP            Type - Source
C                        MOVE 'CUS001  'MODNAM            Source name
C                        MOVE *BLANKS   PLIST             Clear parm list
C                        EXSR OPEN                        Open the member
C*
C              RCODE     IFEQ '0'                         If open successful
C                        MOVE PLIST     PLST,Z             Save parm list
C                        ADD  1         Z                  Bump index
C                        END
C*
C*- Read the first 10 records from each member
C*
C                        MOVE 'N'       OP                Get Next request
C*
C                        MOVE PLST,1    PLIST             Restore proc parm list
C                        Z-ADD1         X
C*
C                        DO   10        X
C                        EXSR NEXT                        Get next record
C                        MOVE TEXT      PROC,X            Save text record
C                        MOVE PLIST     PLST,1            Save parm list
C                        END
C*
C                        MOVE PLST,2    PLIST             Restore source parm list
C                        Z-ADD1         X
C*
C                        DO   10        X
C                        EXSR NEXT                        Get next record
C                        MOVE TEXT      SRC,X             Save text record
```

```
C                   MOVE PLIST     PLIST ?           Save parm list
C                   END
C*
C* Call SUBRSG to open a member
C*
C         OPEN      BEGSR
C                   CALL SUBRSG
C                   RLABL          OP
C                   RLABL          LIBNAM
C                   RLABL          MBRTYP
C                   RLABL          MBRNAM
C                   RLABL          PLIST
C                   RLABL          RCODE
C                   ENDSR
C*
C* Call SUBRSG to get the next list record
C*
C         NEXT      BEGSR
C                   CALL SUBRSG
C                   RLABL          OP
C                   RLABL          PLIST
C                   RLABL          LIST
C                   RLABL          RCODE
C                   ENDSR
C*
```

Re-creating Subroutine SUBRSG

If you don't have assembler subroutine SUBRSG, you can re-create it with procedure MKSUBRSG (you don't need IBM's Assembler Language Program Product to install SUBRSG). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRSG. You need to run MKSUBRSG only once to create the SUBRSG subroutine.

```
// * Re-creating R module SUBRSG in library #RPGLIB
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-301 DATA '00000135'      Number of $MAINT records
// LOCAL OFFSET-208 DATA-'
[illegible]
// LOCAL OFFSET-273 DATA-'
[illegible]
// LOAD MAKMEM
// FILE NAME-BINARY LABEL-$MAINT RETAIN-J BLOCKS-24 EXTEND-24
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT RETAIN-S
// RUN
// COPY FROM-DISK FILE-$MAINT RETAIN-S TO-#RPGLIB
// END
* Patch the new SUBRSG member to insert object code
// LOAD $FEFIX
// RUN
HDR  [illegible] SUBRS00000
PTF  [illegible] RSUBRSG 06  #RPGLIB
DATA [illegible] 00 0000 [illegible]
DATA [illegible] 00 0020 [illegible]
DATA [illegible] 00 0040 [illegible]
DATA [illegible] 00 0060 [illegible]
DATA [illegible] 00 0080 [illegible]
DATA [illegible] 00 00A0 [illegible]
DATA [illegible] 00 00C0 [illegible]
DATA [illegible] 00 00E0 [illegible]
```

Continued

```
DATA 032C 00 0100 E334C0023875C7502058028000113CF281102C28013C0038400116F210043A1001
DATA AE7A 00 0120 78C2020118880900F21C073CF808C3F28706F401040880900002A201C161009
DATA CC41 00 0140 E3330107F210043CF309C3381001287502058C2800013C7502088C770008876
DATA 702C 00 0160 020880000009C30E0101080C0C201000C202000C08700002828221A120A08
DATA C5CA 00 0180 E300C10800000000000000000000000000000000000000000000000000000000
DATA C894 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 982D 00 01C0 E3000114400000000000000000000000000000000000000000000000000000000
DATA 88F6 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
DATA 6495 00 0200 E3118C13C0040404040404040404040000000080084478008001440800000000
DATA FFA2 00 0220 0000000000000000000000000000000000000000000000000000000000001610
DATA BF8F 00 0240 E3080C2000000000C0012000000000000000000000000000000000000000000
DATA 395A 00 0260 0000000000000000000000000000000000000000000000000000000000000000
DATA E080 00 0280 E33AC9FEF2E4C2D9E2C74040040C39697A8888888788A34040803804OF1F8F8F888
DATA 88E3 00 02A0 40C7818 9A84063484092008 1A3A88889404OC1939340998887888A3A240988500
DATA 2895 00 02C0 E3050404A2889 9A585840000000000000000000000000000000000000000000
DATA 4A61 00 02E0 0000000000000000000000000000000000000000000000000000000000000000
DATA 1C74 00 0300 C5000004000000000000000000000000000000000000000000000000000000000
DATA 4444 00 0320 0000000000000000000000000000000000000000000000000000000000000000
DATA 0944 00 0340 8150000000000000000000000000000000000000000000000000000000000000
DATA 6274 00 0360 0000000000000000000000000000000000000000000000000000000000000000
DATA 8844 00 0380 4040404040404040404040404040404040404040404040404040404040404040
DATA 6A14 00 03A0 4040404040404040404040404040404040404040404040404040404040404040
DATA D8E5 00 03C0 4040404040404040404040404040404040404040404040404040404040404040
DATA 1989 00 03E0 4040404040404040404040404040404040404040404040404040404040404040
END  CA11
```

Retrieving Program Source

by Mel Beckman

Code on diskette:
Procedure TRYCMP
RPG program TRYCMP

When your S/36 program needs to read a library source member, you probably use the tried and true technique of invoking the $MAINT utility to copy the source member into a file that your program then processes. Two drawbacks to this method are the extra time required to execute the $MAINT job step and the extra coding required to handle creation and deletion of a job-unique file.

You can circumvent these drawbacks by taking advantage of a little-known and poorly documented SSP facility: the // COMPILE statement. When placed between a // LOAD and // RUN statement for your program, the // COMPILE statement reads a specified source member into a RETAIN-J disk file named $SOURCE. When your program gains control, it simply reads 96 byte source statements from the $SOURCE file. Figure 11-13 shows sample OCL for invoking the // COMPILE function. Note that you must code a // FILE statement for the $SOURCE file, specifying RETAIN-J and an initial records allocation size (i.e., BLOCKS- or RECORDS-). You also should code an EXTEND parameter on the // FILE statement to prevent a "file is full" message if your initial allocation is too small.

Figure 11-14 shows a sample program, TRYCMP, that reads source statements from a source member and prints them. For a program to work with the // COMPILE statement, you must set on the Source Required attribute bit in the program load member.

Figure 11-13

Procedure TRYCMP

```
// LOAD TRYCMP
// FILE NAME-$SOURCE,BLOCKS-10,EXTEND-25,RETAIN-J
// COMPILE INLIB-#LIBRARY,SOURCE-TRYCMP
// RUN
```

Figure 11-14

Progam TRYCMP

```
*       1         2         3         4         5         6         7         8
     H      64                           B                                  TRYCMP
     F$SOURCE IPE F   96  96              DISK
     FPRINT   O   F  132 132              PRINTER
     I$SOURCE NS  01
     I                                           1  96 RECORD
     OPRINT   D          01
     O                          RECORD    96
```

Writing Source and Procedure Members to a Library

by Gary T. Kratzer

Code on diskette:
RPG program TESTSW
Assembler subroutine SUBRSW

To complement subroutine SUBRSG, I now bring you subroutine SUBRSW, which *writes* source and procedure members to a library from an RPG program. Using subroutine SUBRSW is similar to using subroutine SUBRSG (both use the library maintenance routines built into the SSP), but because subroutine SUBRSW creates a library directory entry for the member you are writing, you need to pay close attention to what you're doing.

To use subroutine SUBRSW in an RPG program, you must code an EXIT SUBRSW operation, which must be followed by either two or three RLABL statements, depending on the type of call you are making. The three types of calls — Open, Put Next, and Close — and their respective parameters are described below.

The Open Call

Before you write a new source member, you must make an Open call to subroutine SUBRSW, specifying the directory information — member name, library name, and certain attributes — that you want. An Open call first checks for the existence of the specified library and then sets up the new library directory entry. Subroutine SUBRSW lets you either create new source and procedure members or overwrite existing ones. (If you choose to overwrite an existing member, the text lines you supply will completely replace its contents. You cannot append statements to existing ones in a member.) The format of an Open call is shown in Figure 11-15a, and the three parameter descriptions are as follows:

• OP — one-byte field that contains the operation to be performed. For an Open call, OP should contain the letter O.

• PUTDS — 42-byte data structure that contains detailed information about the member being written. The format of the PUTDS data structure is shown in Figure 11-15b, and a description of the 12 fields contained within it follows:

LIBNAM — Library name to contain the member; left justified.

MEMNAM — Name of the member being created or replaced; left justified.

MEMTYP — Member type. S = source, P = procedure.

RECLNG — Record length of the member. The record length may be from 40 to 120 bytes.

MONUM — Modification reference number.

MODATE — Modification date (YYMMDD).

MOTIME — Modification time (HHMM).

SUBTYP — Member subtype.

LOG — If the member is a procedure and the procedure statements should not be logged to the history file, specify N.

PDATA — If the member is a procedure and data should be passed to the program, specify Y (PDATA-YES).

MRT — If the member is a procedure and should be created as a MRT procedure, specify Y.

DUP — If a duplicate member is to be replaced without a warning message being issued, specify Y.

You should pay close attention to field SUBTYP (i.e., member subtype), which identifies the module as a specific type (e.g., RPG, assembler, screen format). Subtypes are two-digit numbers; the subtype for RPG, for example, is 35. (For additional information on subtype codes and a list of the codes, see the LISTLIBR section in IBM's *System Reference Manual* (SC21-9020).)

• RCODE — one-byte field that contains the return code. The return codes for an Open call are:

0 — Open call successful. Okay to issue Put Next calls.

1 — Library not found.

2 — Library Open failed (library may be corrupt).

The Put Next Call

After successfully opening the member, you make Put Next calls to write individual text records to the member. The format of a Put Next call is shown in Figure 11-16, and the three parameter descriptions are as follows:

• OP — For a Put Next call, OP should contain the letter N.

• TEXT — 120-byte field that contains the next text record to be written to the library. If the member's record length (as specified in RECLNG) is less than 120, you must left-justify the data in field TEXT.

• RCODE — The return codes for a Put Next call are:

0 — Successful Put Next operation.

3 — Library Put Next failed (library may be corrupt).

4 — Library Put Next failed because library is full (you must either condense the library or allocate more space and then call SUBRSW again to try to write the member).

The Close Call

After writing the last text record to the member, you make a close call to subroutine SUBRSW, which closes the library and makes the newly created member available to other users. The format of a Close call is shown in Figure 11-17, and the two parameter descriptions are as follows:

• OP — For a Close call, OP should contain the letter C.

• RCODE — The return codes for a Close call are:

0 — Successful Close operation.

4 — Library Close failed (library may be corrupt).

Using Subroutine SUBRSW

Figure 11-18 shows a short example program, TESTSW, that uses subroutine SUBRSW. Notice that program TESTSW sets the member subtype to 40 (i.e., unspecified) and that the procedure statements should not be logged to the history file.

Subroutine SUBRSW, with subroutine SUBRSG, can be particularly useful when you are writing an editor such as FSEDIT, SEU, or DSU. For example, subroutine SUBRSG can read a member to be edited into a work file, and it also can read in other members so you can include certain statements in the member you're editing. Then subroutine SUBRSW can save the changes by writing the new version back to the library. With these subroutines, you can do all this reading and writing — without the hassle and clumsiness of returning to the calling procedure to invoke $MAINT every time you need to read or write a member.

Figure 11-15a

Calling sequence for subroutine SUBRSW for an open call

```
*        1         2         3         4         5         6         7         8
     C                     EXIT SUBRSW
     C                     RLABL          OP      1         Input
     C                     RLABL          PUTDS             Input
     C                     RLABL          RCODE   1         Output
```

Figure 11-15b

Format of data structure PUTDS

```
*        1         2         3         4         5         6         7         8
     IPUTDS       DS
     I                                        1   8 LIBNAM
     I                                        9  16 MEMNAM
     I                                       17  17 MEMTYP
     I                                       18  20 RECLNG
     I                                       21  26 MONUM
     I                                       27  32 MODATE
     I                                       33  36 MOTIME
     I                                       37  38 SUBTYP
     I                                       39  39 LOG
     I                                       40  40 PDATA
     I                                       41  41 MRT
     I                                       42  42 DUP
```

Figure 11-16

Calling sequence for subroutine SUBRSW for a put next call

```
*        1         2         3         4         5         6         7         8
     C                     EXIT SUBRSW
     C                     RLABL          OP      1         Input
     C                     RLABL          TEXT  120         Input
     C                     RLABL          RCODE   1         Output
```

Figure 11-17

Calling sequence for subroutine SUBRSW for a close call

```
*        1         2         3         4         5         6         7         8
     C                     EXIT SUBRSW
     C                     RLABL          OP      1         Input
     C                     RLABL          RCODE   1         Output
```

Figure 11-18

Sample program TESTSW

```
*        1         2         3         4         5         6         7         8
     H      64                      B                                     TESTSW
     E*
     E                    TXT     1   7 80               Text lines for member
     I*
     IPUTDS       DS
     I                                        1   8 LIBNAM
     I                                        9  16 MEMNAM
     I                                       17  17 MEMTYP
     I                                       18  200RECLNG
     I                                       21  260MONUM
     I                                       27  320MODATE
     I                                       33  360MOTIME
     I                                       37  380SUBTYP
     I                                       39  39 LOG
     I                                       40  40 PDATA
     I                                       41  41 MRT
     I                                       42  42 DUP
     I*
     I            DS
     I                                        1  120TIMDAT
     I                                        1   60TIME
     I                                        1   40HHMM
     I                                        7  120DATE
```

```
I                                      7  100MMDD
I                                     11  120YY
C*
C*- Set attributes
C*
C                    Z-ADD40        SUBTYP            Subtype - Unspecified
C                    MOVE 'N'       LOG               No history logging
C*
C*- Set up the rest of PUTDS
C*
C                    MOVE '#LIBRARY'LIBNAM            Library name
C                    MOVE 'TESTSW  'MEMNAM            Member name
C                    MOVE 'P'       MEMTYP            Type - proc
C                    Z-ADD80        RECLNG            Record length
C                    Z-ADD1         MONUM             Mod number
C                    TIME           TIMDAT            Get current time & date
C                    MOVELYY        MODATE            Mod date - YY
C                    MOVE MMDD      MODATE            Mod date - MMDD
C                    Z-ADDHHMM      MOTIME            Mod time - HHMM
C*
C*- Open the member
C*
C                    MOVE 'O'       OP      1
C                    EXSR OPEN
C*
C*- Write the text records
C*
C                    MOVE 'N'       OP
C                    DO   7         Z      10
C                    MOVELTXT,Z     TEXT  120
C                    EXSR NEXT
C                    END
C*
C*- Close the member
C*
C                    MOVE 'C'       OP
C                    EXSR CLOSE
C*
C          END       TAG
C                    SETON                     LR
C*
C*- Call SUBRSW to open a member
C*
C          OPEN      BEGSR
C                    EXIT SUBRSW
C                    RLABL          OP
C                    RLABL          PUTDS
C                    RLABL          RCODE   1
C                    ENDSR
C*
C*- Call SUBRSW to put the next text record
C*
C          NEXT      BEGSR
C                    EXIT SUBRSW
C                    RLABL          OP
C                    RLABL          TEXT
C                    RLABL          RCODE
C                    ENDSR
C*
C*- Call SUBRSW to close the member
C*
C          CLOSE     BEGSR
C                    EXIT SUBRSW
C                    RLABL          OP
C                    RLABL          RCODE
C                    ENDSR
C*
**
*
*- Dummy procedure to show SUBRSW works
*
// LOAD TESTSW
// FILE NAME-JUNKO,LABEL-?1?,DISP-SHR,DBLOCK-40
// PRINTER NAME-PRINT,FORMSNO-TEST,DEVICE-P2
// RUN
```

Re-creating Subroutine SUBRSW

If you don't have assembler subroutine SUBRSW, you can re-create it with procedure MKSUBRSW (you don't need IBM's Assembler Language Program Product to install SUBRSW). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRSW. You need to run MKSUBRSW only once to create the SUBRSW subroutine.

```
// * 'Re-creating R-module SUBRSW in library #RPGLIB'
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201 DATA-'00000198'      Number of $MAINT records
// LOCAL OFFSET-208 DATA-'
'[illegible]
// LOCAL OFFSET-273 DATA-'
[illegible]
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-F,TO-#RPGLIB
// END
* Patch the new SUBRSW member to insert object code
// LOAD $FEFIX
// RUN
HDR  39C8 SUBRS00000
PTF  0C01 RSUBRSW 00  #RPGLIB
DATA A433 00 0000 [illegible]
DATA 20F3 00 0020 [illegible]
DATA FC00 00 0040 [illegible]
DATA A8F3 00 0060 [illegible]
DATA 0DTA 00 0080 [illegible]
DATA 8131 00 00A0 [illegible]
DATA 3824 00 00C0 [illegible]
DATA 4E80 00 00E0 [illegible]
DATA 8508 00 0100 [illegible]
DATA 1883 00 0120 [illegible]
DATA 0C43 00 0140 [illegible]
DATA 7F84 00 0160 [illegible]
DATA B331 00 0180 [illegible]
DATA 1D62 00 01A0 [illegible]
DATA 0F8A 00 01C0 [illegible]
DATA 0010 00 01E0 [illegible]
DATA F0AF 00 0200 [illegible]
DATA 0CF8 00 0220 [illegible]
DATA 236F 00 0240 [illegible]
DATA 0C53 00 0260 [illegible]
DATA 7071 00 0280 [illegible]
DATA 70A4 00 02A0 [illegible]
DATA 0D25 00 02C0 [illegible]
DATA F1C4 00 02E0 [illegible]
DATA A0EE 00 0300 [illegible]
DATA 6394 00 0320 [illegible]
DATA 8B64 00 0340 [illegible]
DATA E9C5 00 0360 [illegible]
DATA 9F80 00 0380 [illegible]
DATA 9F8F 00 03A0 [illegible]
DATA 8B8D 00 03C0 [illegible]
DATA D98F 00 03E0 [illegible]
DATA E786 00 0400 [illegible]
DATA F8C1 00 0420 [illegible]
DATA E638 00 0440 [illegible]
DATA 5006 00 0460 [illegible]
DATA E499 00 0480 [illegible]
```

Continued

```
DATA E56E 00 04A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 91B8 00 04C0 E30B00B7[illegible]
DATA 8185 00 04E0 0000000000000000000000000000000000000000000000000000000000000000
DATA C786 00 0500 E30000B6[illegible]
DATA 53BF 00 0520 0000000000000000000000000000000000000000000000000000000000000000
DATA 7BF6 00 0540 E33400F2[illegible]
DATA 7829 00 0560 40C7E1[illegible]
DATA 7118 00 0580 E30B00F8[illegible]
DATA 72E0 00 05A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 9606 00 05C0 C50000F8[illegible]
DATA 96E3 00 05E0 0000000000000000000000000000000000000000000000000000000000000000
END  803A
```

Undeleting a Library Member

by Joe Medieros

After hours of frantic work, you've just finished a major program maintenance job on a large S/36 source member called OR1820. It's time to delete the work file from library WORK, but in your haste, you accidentally delete the good copy of the source member. What now? Rather than moving toward a high window, think about how the REMOVE procedure works and how it can be undone.

When a library member is deleted, all that really happens is that certain bytes are reset in the library's directory. To prevent panic in situations such as the one described above, you should know that the member can be restored by setting those bytes to their original configuration. I will explain how the resetting can be done through a directory patch, but first I need to emphasize that the steps I provide have no safeguards. In the hands of a careless, inexperienced, or malicious person, directory byte manipulation can render your system useless.

To "unremove" the source member in the opening example, you must find the starting sector of the library's directory by executing

```
DUMP VTOC,CRT
```

When the dump screen is displayed, roll up until you see the entry for the deleted member's library. The first sector of the directory appears in bytes 33 through 36, which in our example would show 118E76.

Then, with the above-mentioned caveat in mind, and with Service Aid authorization, key in

```
PATCH F1
```

Figure 11-19

Directory sector before patch

```
                         S/36 PATCH DISK UTILITY                          W1
SS@= 0118E77  P              Hexadecimal
  .
  .
  .
   0030    000000E2  D6D9F1F8  F1F54040  00001360    *...SOR1815  ...-*
   0040    003B0000  00000000  00000400  0A200000    *................*
   0050    00158608  06112435  00000080  13660000    *..f........ . ..*
   0060    00000000  0000E2D6  D9F1F8F2  F0404000    *......SOR1820  .*
   0070    01586002  40000000  00000000  00040000    *. -. ..........,*
  .
  .
  .
                                         Sector is patched
Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End
```

Figure 11-20

Directory sector after patch

```
                         S/36 PATCH DISK UTILITY                          W1
SS@= 0118E77  P              Hexadecimal
  .
  .
  .
   0030    000000E2  D6D9F1F8  F1F54040  00001360    *...SOR1815  ...-*
   0040    003B0000  00000000  00000400  0A200000    *................*
   0050    00158608  06112435  00000080  13660000    *..f........ . ..*
   0060    00000000  0000E2D6  D9F1F8F2  F0404000    *......SOR1820  .*
   0070    01586002  40000000  00000000  0004FFFF    *. -. ..........,*
  .
  .
  .
Cmd1-Next sector   Cmd2-Previous sector   Cmd7-End
```

Press the Enter key, and wait for the PATCH screen to appear. Enter the number for the first sector to be displayed. In this example, you would key in 118E76, as determined from the dump screen. After entering the number for the sector to be displayed, press the Field Exit key and then the Enter key, leaving the default parameters as they are. Once the sector is displayed (Figure 11-19), look for the member that you "removed" (in this case, SOR1820, because you removed the source member for program OR1820). If you don't find the member you are looking for in the displayed sector, use Command key 1 to view the next sector(s).

Once you have found the entry that refers to the "removed" member, you can manipulate the appropriate bytes to "unremove" the member. (This is where extreme caution is necessary; you can manipulate any of the bytes in the directory, and an improper manipulation can cause severe problems.) There are 51 bytes in a member's entry, and the first entry is an E2 (the 51 bytes occupy more than one line on the screen). Find the E2 entry on the line that contains the name of the member you "removed." Referring to the E2 as byte number one, count over to byte 25 (remember that each byte consists of two characters). For "removed" members, bytes 25 and 26 are filled with zeros, to indicate that the member no longer exists (see Figure 11-19). To restore the member, replace the zeros with hex FFs by keying FFs into both bytes 25 and 26 and then pressing the Enter key. (Figure 11-20 shows what the directory sector looks like after the FFs have been entered.) Move the cursor to the second field at the top of the PATCH screen, and key a P to indicate that the sector is to be patched. Press Enter. After the "SECTOR IS PATCHED" message appears, use Command key 7 to exit the PATCH procedure. The "removed" member will be back in the library.

Unfortunately, the number of sectors a member occupied cannot be determined by looking at the directory after the member was removed. Keying hex FFs in bytes 25 and 26 will recover the entire member but may also include some extra lines. You will, therefore, need to edit out those extra lines.

Editing the original source member with SEU may not work because certain attribute bytes (such as hex 22) in the extra lines could cause an "ERROR COMMUNICATING WITH DISPLAY STATION" message. You can use the EDIT function of POP, but if you don't have POP, you can use the following steps as a guide:

• Create a new source member, say OR1821, by keying

    SEU OR1821,R,,WORK

• Use Command key 11 to "Include" member OR1820 from library WORK, starting at statement 1.

• Roll up through the "included" source until you find the last good statement of program OR1820. This is the ending statement for your "Include." Key in this statement number, press Enter twice, and the statements are copied into the new member OR1821.

• Remove the old member and use the CHNGEMEM command to change the member name from OR1281 to OR1280.

Re-creating Source from Message and Menu Object Members

by Ron Elliott and Gary T. Kratzer

program by William G. Strejc

Code on diskette:
Procedures CRMSG, CRMENU
RPG program CRSRC
Screen format member CRSRCFM

The RPG library, #RPGLIB, contains assembler subroutine SUBR23, which lets an RPG program retrieve a message from a user message member. Hidden in this subroutine is the "super" ability to convert object code to text. By using this capability, S/36 utility CRSRC can read the object code in a message member or menu and create the corresponding source code member. With utility CRSRC, you can reword a message your manager doesn't like or swap menu items 10 and 14, even if you don't have access to the original source member. What is more, you can modify a menu or message member in a software package supplied *sans* source or for which the original source code has disappeared. And, if "someone" erased the source, this utility could save a life.

Utility CRSRC consists of two procedures, an RPG program, and a screen format member. Procedures CRMSG (to re-create message members) and CRMENU (to re-create menus) request input via prompt screens (Figures 11-21 and 11-22, respectively) and then call program CRSRC. The program calls subroutine SUBR23 and creates an output file for the text.

Figure 11-21
CRMSG prompt screen

```
      Create a Source Message Member From a Level 1 Object Module

Message member name(same as object name)      . .        ________

Library in which the object member resides    .          ________

Record length for the source member   . .       .        080

Print the source member(Y or N) .           . .          Y

Number of messages in the object member           .      0500

            ENTER-Process the screen
            CMD 7-Cancel out of utility
```

Figure 11-22
CRMENU prompt screen

```
Create a Source Menu Message Member From a Level 1 Object Module

Menu message member name(same as menu name).. . .            ________

Library in which the object member resides   ... .  ..       ________

Record length for the source member  ......  ...... ...      120

Print the source member(Y or N) . ...        ...... . ..      Y

Number of messages in the object member...... . .. ...        0024

            ENTER-Process the screen
            CMD 7-Cancel out of utility
```

The procedures then use $COPY to copy the output file into a data file and $MAINT to convert the data file into a source member created in a user-designated library. Both procedures are similar, so we'll describe only how procedure CRMSG works.

Getting Started

Procedure CRMSG (Figure 11-23) begins by displaying the prompt screen (S- and D-specs are included in screen format member CRSRCFM — Figure 11-24). You specify five parameters: the name of the message member, the name of the library, the record length, an option to print the resulting source code, and the number of records the output file will contain. Make sure that this last entry (parameter 5) is at least as large as the highest numbered message in the member. If in doubt, enter 9999 for the fifth parameter value; this choice will waste computer time but will ensure that all messages are retrieved. As shown on the bottom of the prompt screen, Command key 7 cancels the procedure.

Upon return from the prompt screen, procedure CRMSG tests whether you entered a message member name in parameter 1. If not, the procedure is RESET. Next, if the value supplied for the number of messages (parameter 5) is less than two, the default value of 100 is used. Procedure CRMSG then copies the member-name parameter and number-of-messages parameter into the LDA for subsequent use by the RPG program. The procedure sets switch 1 to print or not print the generated source member as specified in parameter 4 and switch 2 to indicate that CRMSG is the calling procedure (procedure CRMENU sets switch 3). The procedure next executes a // MEMBER statement to point to the specified message member in the specified library (the default is the current library). Procedure CRMSG then loads program CRSRC (Figure 11-25).

Building a $MAINT File

Program CRSRC writes a // COPY statement, which is used by $MAINT; writes a header record, which is required in a message source member; calls subroutine SUBR23; writes the text to the output file; and writes a trailing // CEND record, also required by $MAINT.

Program CRSRC begins by picking up the name of the message member from the LDA and then executing subroutine NAME. Subroutine NAME copies the message member name to an array, locates the first blank space in the array, and appends the literal ",1" or ",2", depending on the calling procedure, for output to the message source member header record (message member header records must have the literal ",1" or ",2" following the member name). The header record also contains the literal MSG or MNU (which designates the member subtype), again depending on the switch setting.

Now the program is ready to call subroutine SUBR23. Following the required EXIT statement are four RLABL statements that specify SUBR23's parameters.

The first RLABL specifies a four-digit field that contains the sequence number for the message to be retrieved. Program CRSRC is designed to retrieve all the messages in a member; thus, field MN01 is initialized to 1, and subsequent passes through the program keep incrementing MN01 by 1 until all messages have been retrieved.

The second RLABL specifies an alphabetic field to receive the message. The third RLABL specifies a one-byte field that contains a 1 for first-level message members or a 2 for second-level message members. Because program CRSRC is designed for first-level messages, the field defined in the third RLABL contains a 1.

The final RLABL defines a one-byte field for a return code provided by SUBR23. The return code will be a value of 0 to 5. In brief, a code of 0 or 1 means the desired message was found, and the other return codes mean that it wasn't (for further information, see the description of SUBR23 in the S/36 manual *Programming with RPG II*). After the call to SUBR23, program CRSRC writes the message number and message text to disk file MSGOUT.

No input primary file is declared for program CRSRC, so the RPG cycle routes control back to the beginning of the detail calculations. The message number is incremented, and the retrieval and output processes are repeated until the message number is either greater than field RLIMIT (the number of messages you specified in the prompt screen) or the message number equals zero. If you specified 9999 for the number of messages, the four-byte field MN01 will equal zero on the 10,000th cycle and will set on indicator 50. In either case, the LR indicator comes on, the // CEND trailing record is output, and the program ends.

Before leaving this program, note that the output to the printer file is conditioned on indicator U1. If you opt for printed output, you'll get not only a source listing, but also a printed result of each pass through subroutine SUBR23. As with the disk output, these lines of output are conditioned on the return code from the subroutine and show the message text as well as an informational message.

Saving the Source

When control returns to procedure CRMSG, the $COPY utility copies the file just created (MSGOUT), yielding a new file with a reasonable record length (which you specify in the third parameter on procedure CRMSG's prompt screen — the default record length is 80 bytes). For instance, suppose you know that the maximum length for messages in a particular message member is 40 bytes. Program CRSRC defines the message text field with length 75 bytes. Therefore, in this example, the creation of another copy of the output file saves 35 bytes per record in the final output file, an important consideration when you realize some system message members contain thousands of messages. The copy process also renames the file so that it now exists on the disk under the label specified as the first procedural parameter.

Finally, the call to $MAINT in procedure CRMSG creates, from the disk file, a source member in the library specified by the second parameter. When procedure CRMSG is completed, the desired source for the specified message member exists in the library you specified as parameter 2. You can then modify the messages as you see fit.

Remember that program CRSRC can build a source member for menus, too. Simply call procedure CRMENU (Figure 11-26), which in turn displays the prompt screen similar to the one procedure CRTMSG displays. As required for menu members, procedure CRMENU appends the literal ## to the first parameter value.

With utility CRSRC, you can change message text and menu wordings and selections even if you do not have the source code — adding to your reputation as a "can do" programmer.

Figure 11-23
Procedure CRMSG

```
**
**  CRMSG - CREATE A SOURCE MESSAGE MEMBER FROM AN
**          OBJECT MESSAGE MEMBER.
**
// PROMPT MEMBER-CRSRCFM,FORMAT-WSSO33A,START-1,LENGTH-'8,8,3,1,4'
// IF ?CD?=2007 RETURN
// IF ?1?=  RESET CRMSG
// IF 2>?5? EVALUATE P5,4=0100
// LOCAL OFFSET-1,DATA-'?1?
// LOCAL OFFSET-9,DATA-'?5?
// SWITCH 010XXXXX
// IF ?4?=Y SWITCH 1XXXXXXX
// MEMBER USER1-?1?,LIBRARY-?2'?CLIB?'?
// IF DATAF1-MSGOUT DELETE MSGOUT,F1
// IF DATAF1-?1?    DELETE ?1?,F1
// LOAD CRSRC
// FILE NAME-MSGOUT,RECORDS-?5?,EXTEND-50
```

```
// RUN
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-MSGOUT,RETAIN-S
// FILE NAME-COPYO,LABEL-?1?,RECORDS-?F'A,MSGOUT'?
// RUN
// COPYFILE OUTPUT-SAME,RECL-?3'80'?
// END
// LOAD $MAINT
// FILE NAME-?1?,UNIT-F1
// RUN
// COPY FROM-DISK,TO-?2?,RETAIN-P,FILE-?1?
// END                          .
// SWITCH 000XXXXX
// IF DATAF1-MSGOUT DELETE MSGOUT,F1
// IF DATAF1-?1?    DELETE ?1?,F1
```

Figure 11-24

Screen format member CRSRCFM

```
*       1  .  ... 2 ..     3    .  4        5         6         7         8
0001 SWSS033A
0002 D          59 211Y                                  CCreate a Source MessageX
0003 D Member From a Level 1 Object Module
0004 D          54 5 7Y                                  CMessage member name(samX
0005 De as object name) .     ..  . ..
0006 DMNAME      8 566Y  Y            Y         Y
0007 D          54 7 7Y                                  CLibrary in which the obX
0008 Dject member resides    . ....
0009 DLNAME      8 766Y  Y            Y         Y
0010 D          54 9 7Y                                  CRecord length for the sX
0011 Dource member        .. .    .
0012 DRLEN       3 966Y  YN   B       Y         Y        C080
0013 D          5411 7Y                                  CPrint the source memberX
0014 D(Y or N)     .. .       .
0015 DYORN      11166Y  Y            Y         Y        CY
0016 D          5413 7Y                                  CNumber of messages in tX
0017 Dhe object member          ..
0018 DRLIMIT    41366Y  YN   Z       Y         Y        C0500
0019 D          241519Y                                  CENTER-Process the screeX
0020 Dn
0021 D          271619Y                                  CCMD 7-Cancel out of utiX
0022 Dlity
0023 D          4024 7Y                    Y             CCopyright 1986 by WilliX
0024 Dam G  Strejc Inc
0025 SWSS033B
0026 D          64 2 7Y                                  CCreate a Source Menu MeX
0027 Dssage Member From a Level 1 Object Module
0028 D          54 5 7Y                                  CMenu message member namX
0029 De(same as menu name)   .    ..
0030 DMNAME      8 566Y  Y            Y         Y
0031 D          54 7 7Y                                  CLibrary in which the obX
0032 Dject member resides
0033 DLNAME      8 766Y  Y            Y         Y
0034 D          54 9 7Y                                  CRecord length for the sX
0035 Dource member       . .     .
0036 DRLEN       3 966Y  YN   B       Y         Y        C120
0037 D          5411 7Y                                  CPrint the source memberX
0038 D(Y or N)           . .
0039 DYORN      11166Y  Y            Y         Y        CY
0040 D          5413 7Y                                  CNumber of messages in tX
0041 Dhe object member.        .   .
0042 DRLIMIT    41366Y  YN   Z       Y         Y        C0024
0043 D          241519Y                                  CENTER-Process the screeX
0044 Dn
0045 D          271619Y                                  CCMD 7-Cancel out of utiX
0046 Dlity
0047 D          4024 7Y                    Y             CCopyright 1986 by WilliX
0048 Dam G. Strejc Inc
```

Figure 11-25

Program CRSRC

```
•   1         2         3         4         5         6         7         8
H      64                      8       1                             CRSRC
F*****************************************************
F*
F*  PROG NAME   CRSRC
F*  PROG DESC   CREATE A MESSAGE MEMBER OR MENU SOURCE
F*              FILE FROM AN OBJECT MEMBER & PRINT THE
F*              MESSAGE OR MENU TEXT
F*  AUTHOR      WILLIAM STREJC
F*
FPRINT   O   F 132 132    2       PRINTER
FMSGOUT  O   F 120 120    2       DISK
E                    NA          10  1               MESSAGE MEMBER NAME ARRAY
I           UDS
I                                        1   8 MN                    MESSAGE MEMBER NAME
I                                        9  12ORLIMIT                MESSAGE # LIMIT
C           FIRST     IFNE 'Y'
C                     Z-ADD1         MLVL    10
C                     EXSR NAME
C                     EXCPTCOPY1                      OUTPUT COPY STATEMENT
C                     EXCPTHED1                       PRINT & OUTPUT HEADER RECORD
C                     MOVE 'Y'       FIRST   1
C                     END
C                     ADD  1         MNO1    40
C           MNO1      COMP RLIMIT               50
C  N50      MNO1      COMP *ZEROS                   50
C   50                SETON                     LR
C   50                EXCPTCEND1                      OUTPUT CEND STATEMENT
C   50                GOTO END                        END OF READ AND DISPLAY LOOP
C                     EXIT SUBR23
C                     RLABL          MNO1             FOUR DIGIT FIELD
C                     RLABL          MCMD    75       TEXT FOR THIS MESSAGE MEMBER
C                     RLABL          MLVL             MESSAGE LEVEL(1 OR 2)
C                     RLABL          MRCD    1        MESSAGE NUMBER(RETURN CODE)
C                     MOVE MCMD      M75     75
C           MRCD      COMP '0'                    10 MSG RETRIEVED. NO TRUNCATION
C           MRCD      COMP '1'                    11 MSG RETRIEVED. BUT TRUNCATED
C           MRCD      COMP '2'                    12 MSG NOT FOUND
C           MRCD      COMP '3'                    13 MSG LEVEL IS INVALID
C           MRCD      COMP '4'                    14 INVALID MIC VALUE DIAGNOSED
C           MRCD      COMP '5'                    15 MSG NOT FOUND OR LENGTH EXCEEDS
C*                                                   LEVEL-1 MAXIMUM
C  N12                EXCPTPRNT1                      PRINT MSG & OUTPUT A RECORD
C           END       TAG
C*****************************************************
C*  SUBROUTINE NAME - SET UP THE MEMBER NAME ARRAY
C           NAME      BEGSR
C                     MOVEAMN        NA
C                     Z-ADD1         M       20
C           STAG1     TAG
C                     ADD  1         M
C           M         COMP 9                      50
C  N50      NA,M      COMP                        50
C  N50                GOTO STAG1
C   U2                MOVEA'.1'      NA,M             MESSAGE
C   U3                MOVEA'.2'      NA,M             MENU
C                     ENDSR
OPRINT   H  203     1P U1
D                                   30 'MESSAGE LISTING AS OF'
O                         UDATE Y   39
O        H  1       1P U1
O                                    4  L C'
O        H  1       1P U1
O                                    4 'V D'
O        H  2       1P U1
O                                    4 'L E'
O                                   13 'MSG#
O                                   23  TEXT FOR'
O                         MN        32
O        E  1       U1    PRNT1
```

```
 O                             MLVL       2
 O                             MRCD       4
 O                             MNO1      13
 O                             M75       90
 O                        10            132 'MSG FOUND, COMPLETE'
 O                        11            132 'MSG FOUND, TRUNCATED'
 O                        13            132 'MESSAGE LEVEL INVALID'
 O                        14            132 'INVALID MIC VALUE'
 O                        15            132 'LENGTH EXCEEDS LVL-1 MAX'
 OMSGOUT  E                    HED1
 O                             NA        10
 O        E                    PRNT1
 O                             MNO1       4
 O                             M75       80
 O        E                    COPY1
 O                                       24 '// COPY  LIBRARY-S,SUB-M'
 O                   U2                  32 'SG,NAME-'
 O                   U3                  32 'NU,NAME-'
 O                             MN        40
 O        E                    CEND1
 O                                        7 '// CEND'
```

Figure 11-26

Procedure CRMENU

```
**
**  CRMENU - CREATE A SOURCE MENU MESSAGE MEMBER FROM AN
**           OBJECT MENU MESSAGE MEMBER.
**
// PROMPT MEMBER-CRSRCFM,FORMAT-WSS033B,START-1,LENGTH-'8,8,3,1,4'
// IF ?CD?=2007 RETURN
// IF ?1?=  RESET CRMENU
// IF 2>?5? EVALUATE P5,4=0100
// LOCAL OFFSET-1,DATA-'?1?##      '
// LOCAL OFFSET-9,DATA-'?5?   '
// SWITCH 001XXXXX
// IF ?4?=Y SWITCH 1XXXXXXX
// MEMBER USER1-?1?##,LIBRARY-?2'?CLIB?'?
// IF DATAF1-MSGOUT DELETE MSGOUT,F1
// IF DATAF1-?1?##  DELETE ?1?##,F1
// LOAD CRSRC
// FILE NAME-MSGOUT,RECORDS-?5?,EXTEND-50
// RUN
// LOAD $COPY
// FILE NAME-COPYIN,LABEL-MSGOUT,RETAIN-S
// FILE NAME-COPYO,LABEL-?1?##,RECORDS-?F'A,MSGOUT'?
// RUN
// COPYFILE OUTPUT-SAME,RECL-?3'120'?
// END
// LOAD $MAINT
// FILE NAME-?1?##,UNIT-F1
// RUN
// COPY FROM-DISK,TO-?2?,RETAIN-P,FILE-?1?##
// END
// SWITCH 000XXXXX
// IF DATAF1-MSGOUT DELETE MSGOUT,F1
// IF DATAF1-?1?##  DELETE ?1?##,F1
```

Re-creating Source from Format, Menu, and Message Object Members

by Mel Beckman

As a S/36 user, you may find a portion of the AS/400 migration aid (feature 5272MG1) useful even if you aren't migrating to the AS/400. The S/36 half of the migration aid contains three procedure commands — FMT2SRC, MNU2SRC, and MSG2SRC — that convert S/36 format members, menu load members, and message load members, respectively, back into the source code used to create them originally. Each utility can convert either a single load member or all the load members in one library.

You could use these utilities to recover lost source code for your own application load members or to extract the source code from program products (either IBM or third-party vendors) that don't supply source for screens, menus, or messages. Once retrieved, you easily can modify the source code (with SDA or a text editor) and recompile it to create new load members customized for your own needs. If your third-party accounting package doesn't allow lowercase input of names and addresses, for example, you easily could retrieve, modify, and recompile the affected screen formats to permit lowercase entry. Non-English-speaking users likewise could translate screen formats, menus, and messages into their native languages to make third-party applications user-hospitable.

Setting Library Member Attributes

by Gary T. Kratzer

program by Mel Beckman

Code on diskette:
Procedure ATRSET
RPG program ATRSET
Screen format member ATRSETFM

Do you want to execute a program on your S/36 without the fear of other programs getting in the way? Do you want to restrict a particular program to run from the system console only? How about changing a library member's subtype so POP's auto-recognition feature prompts you for the correct com-

Figure 11-27
Parameter prompt screen

```
                        Library Directory Entry
                         Attribute Set Utility

Module Name  .          .    .       .   .  SPLARC
Module Type (O,R,P,S)        .       .  . . . . . 0
Library Name       .         .       .      GARY







                           Cmd7-Cancel            Enter-Proceed
```

piler after editing a program? These capabilities and more can be yours with utility ATRSET (library member attribute set utility).

On-line utility ATRSET (see Figure 11-27 for the parameter prompt screen) lets you alter any library member's attributes or directory information, eliminating the need to write a unique or quick-and-dirty program each time you need to change an attribute. Library attributes are bits of information associated with a library member (i.e., object, subroutine, procedure, or source) that influence the way SSP processes the library member. For example, the NOLOG attribute of procedure members tells SSP whether to log a procedure's statements to the history file.

You create utility ATRSET by creating procedure ATRSET (Figure 11-28) and by compiling program ATRSET (Figure 11-29) and screen format member ATRSETFM (Figure 11-30). Call procedure ATRSET to activate the program; three parameters are required. If you don't key the parameters on the procedure line, the prompt screen requests them. The first parameter is the name of the module you want to change, the second parameter is the module type (i.e., O for Object, R for Subroutine, P for Procedure, S for Source), and the third parameter is the library in which the member resides.

If the member you're changing exists, it is copied via IBM's utility $MAINT to an eight-byte-record work file called MODFILE, which is defined as RETAIN-J. Although the entire member is copied to the work file, program ATRSET uses only the first seven records because they hold the directory information. Even though program ATRSET doesn't verify explicitly the existence of the member or library you specify, you will receive an error message if the member or library doesn't exist when $MAINT tries to copy it to the work file.

After creating the work file, program ATRSET displays the member's first three attribute bytes (Figure 11-31); the corresponding bit status and a brief description accompany each attribute. Digit 1 indicates the bit is on, and 0 indicates it is off. If you want to change the attributes, simply key 1 or 0 over the existing value. For a more detailed description of library attributes, see the *IBM System Reference Manual* (SC21-9020).

Press Enter again to see attribute bytes four, five, and six on the screen (Figure 11-32). This screen is similar to the one described above except for one minor difference; instead of displaying the eight bits that can be set individually as in attribute bytes one, two, and three, attribute byte five specifies a two-digit member subtype assigned to the module. If you want to change the existing subtype, simply key over it the corresponding two-digit subtype you wish to assign the member. The valid subtypes and their values are provided on the screen.

Press Enter again, and the screen in Figure 11-33 is displayed. This screen contains other miscellaneous fields that reside in the member's directory entry. These fields include the MRTMAX count (for O-modules), release level, reference number, and date and time the member was last modified. You can change these values by keying over the existing data.

If you want to review your changes at any time during the process, press Command key 2 to scroll back through the entry screens. Press Enter to update the member and copy it back into the library. If you decide you do not want to update the member, press Command key 7 to cancel the procedure.

One application for utility ATRSET is particularly useful if you own a S/36 5363. Bit 2 (emulation member) in attribute 6 is on for all object and subroutine members; when you attempt to move the module to a different model S/36 (5360, 5362, or 5364), you get an error message saying that the system cannot copy this member (i.e., error message SYS-2462: *module name* Cannot copy this member). IBM apparently does not want library members to be traded from the smaller, less expensive S/36 to the larger models; you can, however, transfer members to other machines safely. Simply use program ATRSET to turn the bit off.

Another useful trick for those of you with any S/36 model is to set bit 0 on in attribute byte 2 (dedicated module) of any load member. When this bit is on, the program can be run only if no other jobs are running on the system. Likewise, once your dedicated program is running, no other jobs may be initiated. This technique is quite handy for shops that have trouble keeping users off the system when dedication is required.

By examining the various library attribute bits, I'm sure you can come up with many other uses for this utility. Forget about quick-and-dirty programs each time you need to change a library member's attribute. Instead, clean up your act — pull utility ATRSET out of your programming arsenal to get the job done quickly but cleanly.

Figure 11-28

Procedure ATRSET

```
*
* System/36 library member attribute set utility
*
* Set defaults
// EVALUATE P2-?2'0'?
// EVALUATE P3-?3'?SLIB?'?
* Check execution environment
// IFF SECURITY-S RETURN                 Don't run if not authorized
// IF EVOKED-NO IF JOBO-NO IFF ?4?/  * 'Attribute set utility is running'
* If parm 1 missing, prompt for first three parms
// IF ?1?/ PROMPT MEMBER-ATRSETFM,FORMAT-ATRSETOO
// IF ?CD?/2007 RETURN  Quit on Cmd7
* Set primary LDA values
// LOCAL OFFSET-451,DATA-'?1?',BLANK-8
// LOCAL OFFSET-459,DATA-'?2?',BLANK-1
// LOCAL OFFSET-460,DATA-'?3?',BLANK-8
* Copy library member to a sector mode file
// LOAD $MAINT
// FILE NAME-MOOCOPY,BLOCKS-50,EXTENO-100,RETAIN-J
// RUN
// COPY FROM-?3?,LIBRARY-?2?,TO-DISK,FILE-MOOCOPY,NAME-?1?
// END
* Select batch or interactive mode
// IF ?4?/ SWITCH 01XXXXXX     If parm 4 is missing, then set interactive mode
// ELSE    SWITCH 00XXXXXX     Otherwise set batch mode
* Set up LDA parameters for batch mode
// IF SWITCH2-1 GOTO NOTBATCH
// LOCAL OFFSET-468,DATA-'?4?',BLANK-8
// LOCAL OFFSET-476,DATA-'?5?',BLANK-8
```

```
// LOCAL OFFSET-484,DATA-'?6?',BLANK-8
// LOCAL OFFSET-492,DATA-'?7?',BLANK-8
// LOCAL OFFSET-500,DATA-'?8?',BLANK-2
// LOCAL OFFSET-502,DATA-'?9?',BLANK-8
// TAG NOTBATCH
* Run attribute set program
// LOAD ATRSET
// FILE NAME-MODCOPY
// RUN
// IF SWITCH1-1 RETURN    If Cmd7 pressed in interactive mode, get out
* Replace module in library
// LOAD $MAINT
// FILE NAME-MODCOPY
// RUN
// COPY FROM-DISK,FILE-MODCOPY,TO-?3?,RETAIN-R
// END
```

Figure 11-29

Program ATRSET

```
* .  .    1         2         3 .       4         5         6    .    7         8
H    014                           B       1                             ATRSET
F*************************************************************************
F*                                                                       *
F*  Copyright 1984, 1985 by Mel Beckman                                  *
F*                                                                       *
F*      Name  ATRSET - System/36 Attribute Set Utility                   *
F*   Created  12/01/83                                                   *
F*    Author  Mel Beckman                                                *
F*   Version  1 1                                                        *
F*                                                                       *
F*************************************************************************
F*                                                                       *
F*   This program reads a $MAINT sector mode file containing a library  *
F*   member and sets the attributes according to user instructions from *
F*   the LDA or workstation                                              *
F*                                                                       *
F*   U1      is set *ON if the user requests EOJ (CMD-7) anytime during  *
F*           interactive mode  This tells the procedure to quit          *
F*                                                                       *
F*   U2 *ON  selects interactive mode  Directory attributes are dis-     *
F*           played and optionally updated by the user                   *
F*                                                                       *
F*   U2 *OFF selects batch mode.  Attribute byte changes are read from   *
F*           the LDA and used to update the library file                 *
F*                                                                       *
F*   After this program ends, $MAINT copies the sector mode file back to *
F*   the originating library                                             *
F*                                                                       *
F*************************************************************************
F/EJECT
F@WORKSTNCD          64            WORKSTN                        U2
FMODCOPY UC           8R           DISK
E*
E* Array of 8-byte directory entry chunks
E*
E                    DIR         6  8
E*
E* Arrey containing attribute bits for one byte
E*
E                    A           8  1
E*
E* Arrays for converting to and from hex
E*
E                    DIG        16  1   VAL     1
E*
E* Ordinal bit value array, bits 0, 1, 2, 3, 4, 5, 6, 7
E*
E                    ORD         8  1
I/EJECT
I*
I* Screen 01
I*
I@WORKSTN         1 CO   2 C1
I                                          3  10 MASK1
I                                         11  18 MASK2
I                                         19  26 MASK3
```

```
I*
I* Screen 02
I*
I@WORKSTN         1 CO   2 C2
I                                          3  10 MASK4
I                                         11  12 MASK5
I                                         13  20 MASK6
I*
I* Screen 03
I*
I@WORKSTN         1 CO   2 C3
I                                          3   4 XMRT
I                                          5   6 XREL
I                                          7  12 XMOD
I                                          7   8 XMOD1
I                                          9  10 XMOD2
I                                         11  12 XMOD3
I                                         13  18 XDATE
I                                         13  14 XDATE1
I                                         15  16 XDATE2
I                                         17  18 XDATE3
I                                         19  22 XTIME
I                                         19  20 XTIME1
I                                         21  22 XTIME2
I/EJECT
I*
I* Attribute byte record input
I*
IMODCOPY
I                                          1   8 DIR,X
I*
I* Directory record data structure   (only first 40 bytes are needed)
I*
I            DS
I                                          1  48 DIR
I                                         20  20 ATTR1
I                                         21  21 ATTR2
I                                         22  22 ATTR3
I                                         23  23 MRT
I                                         24  24 REL
I                                         27  27 ATTR4
I                                         29  29 MOD1
I                                         30  30 MOD2
I                                         31  31 MOD3
I                                         32  32 DATE1
I                                         33  33 DATE2
I                                         34  34 DATE3
I                                         35  35 TIME1
I                                         36  36 TIME2
I                                         37  37 ATTR5
I                                         40  40 ATTR6
I*
I* Local data area contains bit and byte masks for batch mode
I*
I           UDS
I                                        451 458 MODNAM
I                                        459 459 TYPE
I                                        460 467 LIBNAM
I                                        468 475 MASK1
I                                        476 483 MASK2
I                                        484 491 MASK3
I                                        492 499 MASK4
I                                        500 501 MASK5
I                                        502 509 MASK6
I/EJECT
C*
C* Initialize hex conversion tables
C*
C                     BITOF'01234567'XOO     1          Constant X'00'
C                     MOVE XOO       VAL                Clear hex values  A
C                     BITON'7'       VAL,2              X'01'
C                     BITON'6'       VAL,3              X'02'
C                     BITON'67'      VAL,4              X'03'
C                     BITON'5'       VAL,5              and on
C                     BITON'57'      VAL,6              and on
C                     BITON'56'      VAL,7              ad nauseum
```

```
C                     BITON'567'     VAL,8
C                     BITON'4'       VAL,9
C                     BITON'47'      VAL,10
C                     BITON'46'      VAL,11
C                     BITON'467'     VAL,12
C                     BITON'45'      VAL,13
C                     BITON'457'     VAL,14
C                     BITON'456'     VAL,15
C                     BITON'4567'    VAL,16
C*
C                     MOVEA'01234567'DIG,1            Hex digits from A-F
C                     MOVEA'89A8CDEF'DIG,9
C*
C                     MOVE X00       ORD              Clear to X'00'
C                     BITON'0'       ORD,1            Bit 0
C                     BITON'1'       ORD,2                1
C                     BITON'2'       ORD,3                2
C                     BITON'3'       ORD 4                3
C                     BITON'4'       ORD,5                4
C                     BITON'5'       ORD,6                5
C                     BITON'6'       ORD,7                6
C                     BITON'7'       ORD,8                7
C*
C* Define local variables
C*
C                     MOVE *BLANKS   ATTR    1
C                     MOVE *BLANKS   BYTE    1
C                     MOVE *BLANKS   X      20
C                     MOVE *BLANKS   HEXDIG  2
C                     MOVE *BLANKS   HEX1    1
C                     MOVE *BLANKS   HEX2    1
C                     MOVE *BLANKS   BITS    1
C                     MOVE *BLANKS   HOLD8   8
C*
C* Output heading display if interactive mode
C*
C   U2                SETON                     01    Protect input flds
C   U2                EXCPTSCRN00
C*
C* Get the directory entry from the first six 8-byte records in MODCOPY
C*
C                     DO   6         X      20        Do 6 times
C          X          CHAINMODCOPY                     Get dir chunk
C                     END                             SAVE INFO
C*
C* If U2 is off, this is batch mode  Go directly to update logic
C*
C  NU2                GOTO UPDATE
C/EJECT
C*
C* Interactive update of attributes
C*
C*     Call the format routine to convert binary and hex to screen
C*     representation
C*     Display the screens, let the user update
C*     Fall through to the update routine to update the records
C*
C*
C                     EXSR FORMAT                     Format for display
C*
C          SHOW01     TAG
C                     EXCPTSCRN01                     Show screen 1
C                     READ @WORKSTN               1111 Read it
C   KG                SETON                     U1    If EOJ,
C   KG                GOTO END                         Then quit
C*
C          SHOW02     TAG
C                     EXCPTSCRN02                     Show screen 2
C                     READ @WORKSTN               1111 Read it
C   KB                GOTO SHOW01                     If backup, go back
C   KG                SETON                     U1    If EOJ,
C   KG                GOTO END                         Then quit
C*
C          SHOW03     TAG
C                     EXCPTSCRN03                     Show screen 3
C                     READ @WORKSTN               1111 Read it
```

```
C   KB              GOTO SHOWO2                      If backup, go back
C   KG              SETON                       U1   If EOJ,
C   KG              GOTO END                          Then quit
C/EJECT
C*
C* Update the attribute bits and bytes from the input bit and byte masks
C*
C          UPDATE   TAG
C*
C          MASK1    IFNE *BLANKS                     If mask not blank
                    MOVEAMASK1     A,1                Get bitmask
C                   MOVE ATTR1     ATTR               And attribute byte
C                   EXSR BITSET                       Set bits
C                   MOVE ATTR      ATTR1              Restore attribute
C                   END                              End IF
C*
C          MASK2    IFNE *BLANKS                     If mask not blank
C                   MOVEAMASK2     A,1                Get bitmask
C                   MOVE ATTR2     ATTR               And attribute byte
C                   EXSR BITSET                       Set bits
C                   MOVE ATTR      ATTR2              Restore attribute
C                   END                              End IF
C*
C          MASK3    IFNE *BLANKS                     If mask not blank
C                   MOVEAMASK3     A,1                Get bitmask
C                   MOVE ATTR3     ATTR               And attribute byte
C                   EXSR BITSET                       Set bits
C                   MOVE ATTR      ATTR3              Restore attribute
C                   END                              End IF
C*
C          MASK4    IFNE *BLANKS                     If mask not blank
C                   MOVEAMASK4     A,1                Get bitmask
C                   MOVE ATTR4     ATTR               And attribute byte
C                   EXSR BITSET                       Set bits
C                   MOVE ATTR      ATTR4              Restore attribute
C                   END                              End IF
C*
C          MASK5    IFNE *BLANKS                     If mask not blank
C                   MOVE MASK5     HEXDIG             Get byte mask
C                   EXSR BYTSET                       Set byte
C                   MOVE BYTE      ATTR5              Restore byte
C                   END                              End IF
C*
C          MASK6    IFNE *BLANKS                     If mask not blank
C                   MOVEAMASK6     A,1                Get bitmask
C                   MOVE ATTR6     ATTR               And attribute byte
C                   EXSR BITSET                       Set bits
C                   MOVE ATTR      ATTR6              Restore attribute
C                   END                              End IF
C*
C          XMRT     IFNE *BLANKS                     If mask not blank
C                   MOVE XMRT      HEXDIG             Get byte mask
C                   EXSR BYTSET                       Set byte
C                   MOVE BYTE      MRT                Restore byte
C                   END                              End IF
C*
C          XREL     IFNE *BLANKS                     If mask not blank
C                   MOVE XREL      HEXDIG             Get byte mask
C                   EXSR BYTSET                       Set byte
C                   MOVE BYTE      REL                Restore byte
C                   END                              End IF
C*
C          XMOD     IFNE *BLANKS                     If mask not blank
C                   MOVE XMOD1     HEXDIG             Get byte mask
C                   EXSR BYTSET                       Set byte
C                   MOVE BYTE      MOD1               Restore byte
C                   MOVE XMOD2     HEXDIG             Get byte mask
C                   EXSR BYTSET                       Set byte
C                   MOVE BYTE      MOD2               Restore byte
C                   MOVE XMOD3     HEXDIG             Get byte mask
C                   EXSR BYTSET                       Set byte
C                   MOVE BYTE      MOD3               Restore byte
C                   END                              End IF
C*
C          XDATE    IFNE *BLANKS                     If mask not blank
C                   MOVE XDATE1    HEXDIG             Get byte mask
```

```
C                     EXSR BYTSET                       Set byte
C                     MOVE BYTE      DATE1              Restore byte
C                     MOVE XDATE2    HEXDIG             Get byte mask
C                     EXSR BYTSET                       Set byte
C                     MOVE BYTE      DATE2              Restore byte
C                     MOVE XDATE3    HEXDIG             Get byte mask
C                     EXSR BYTSET                       Set byte
C                     MOVE BYTE      DATE3              Restore byte
C                     END                              End IF
C*
C           XTIME     IFNE *BLANKS                     If mask not blank
C                     MOVE XTIME1    HEXDIG             Get byte mask
C                     EXSR BYTSET                       Set byte
C                     MOVE BYTE      TIME1              Restore byte
C                     MOVE XTIME2    HEXDIG             Get byte mask
C                     EXSR BYTSET                       Set byte
C                     MOVE BYTE      TIME2              Restore byte
C                     END                              End IF
C/EJECT
C*
C* Update the attribute records
C*
C                     DO   6         X                 Do 6 times
C                     MOVE DIR,X     HOLD8              Save dir chunk
C           X         CHAINMODCOPY                      Get from file
C                     MOVE HOLD8     DIR,X              Update the chunk
C                     EXCPTUPDREC                       Update the record
C                     END                              End IF
C/SPACE 3
C*
C* End of program
C*
C           END       TAG
C                     SETON                      LR    FORCE EOJ
C/EJECT
C*
C* FORMAT routine   Convert directory fields to displayable form
C*
C*   Input- The directory records
C*  Output- Directory fields converted to displayable form
C*
C*
C           FORMAT    BEGSR
C*
C                     MOVE ATTR1     ATTR              GET ATTRIBUTE BYTE
C                     EXSR BITSHO                      CALL BIT DISPLAY RO
C                     MOVEAA,1       MASK1             AND COPY OUT MASK
C*
C                     MOVE ATTR2     ATTR              GET ATTRIBUTE BYTE
C                     EXSR BITSHO                      CALL BIT DISPLAY RO
C                     MOVEAA,1       MASK2             AND COPY OUT MASK
C*
C                     MOVE ATTR3     ATTR              GET ATTRIBUTE BYTE
C                     EXSR BITSHO                      CALL BIT DISPLAY RO
C                     MOVEAA,1       MASK3             AND COPY OUT MASK
C*
C                     MOVE ATTR4     ATTR              GET ATTRIBUTE BYTE
C                     EXSR BITSHO                      CALL BIT DISPLAY RO
C                     MOVEAA,1       MASK4             AND COPY OUT MASK
C*
C                     MOVE ATTR5     BYTE              GET ATTRIBUTE BYTE
C                     EXSR BYTSHO                      CALL BIT DISPLAY RO
C                     MOVE HEXDIG    MASK5             AND COPY OUT MASK
C*
C                     MOVE ATTR6     ATTR              GET ATTRIBUTE BYTE
C                     EXSR BITSHO                      CALL BIT DISPLAY RO
C                     MOVEAA,1       MASK6             AND COPY OUT MASK
C*
C                     MOVE MRT       BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XMRT              AND COPY OUT 2 HEX
C*
C                     MOVE REL       BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XREL              AND COPY OUT 2 HEX
C*
```

```
C                     MOVE MOD1      BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XMOD1             AND COPY OUT 2 HEX
C                     MOVE MOD2      BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XMOD2             AND COPY OUT 2 HEX
C                     MOVE MOD3      BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XMOD3             AND COPY OUT 2 HEX
C*
C                     MOVE DATE1     BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XDATE1            AND COPY OUT 2 HEX
C                     MOVE DATE2     BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XDATE2            AND COPY OUT 2 HEX
C                     MOVE DATE3     BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XDATE3            AND COPY OUT 2 HEX
C*
C                     MOVE TIME1     BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XTIME1            AND COPY OUT 2 HEX
C                     MOVE TIME2     BYTE              GET BYTE
C                     EXSR BYTSHO                      CALL BYTE DISPLAY R
C                     MOVE HEXDIG    XTIME2            AND COPY OUT 2 HEX
C*
C                     ENDSR
C/EJECT
C*
C*
C* BITSET routine.  Set attribute bit values
C*
C*  Input- A     is an 8-byte array of mask characters
C*               1 means seton, 0 means setoff, X means leave alone
C*         ATTR  is the one-byte field to be set with the mask
C* Output- ATTR  is set according to the mask
C*
C           BITSET    BEGSR
C*
C                     DO   8         X                 Do for 8 bits
C           A,X       IFEQ '0'                          If user set on
C                     BITOFORD,X     ATTR                Set bit on
C                     END                               End IF
C           A,X       IFEQ '1'                          If user set off
C                     BITONORD,X     ATTR                Set bit off
C                     END                               End IF
C                     END                              End DO
C*
C                     ENDSR
C/EJECT
C*
C* BITSHO routine   Make attribute bits displayable
C*
C*  Input- ATTR  Is the byte to be displayed
C* Output- A     Is an 8-byte array of mask characters
C*
C           BITSHO    BEGSR
C*
C                     DO   8         X                 Do for 8 bits
C                     TESTBORD,X     ATTR           11  Test bit
C   11                MOVE '1'       A,X                 If on, mask=1
C  N11                MOVE '0'       A,X                 If off, mask=0
C                     END                              End DO
C*
C                     ENDSR
C/EJECT
C*
C* BYTSET routine   Convert hex to binary
C*
C*  Converts literal hex values in the HEXDIG field to binary data
C*  Characters are translated only if HEXDIG contents are not blank
C*
C*  Input- HEXDIG is a 2-character field containing the hex digits for
C*                the byte
C* Output- BYTE   Is the byte.
```

```
C*
C           BYTSET    BEGSR
C*
C           HEXDIG    IFNE *BLANKS                        If not blank
C                     Z-ADD1         X
C                     MOVE HEXDIG    HEX1
C           HEX1      LOKUPDIG,X                       11  Lookup r.h  digit
C  N11                MOVE VAL,1     HEX1                    (If bad, use 0)
C   11                MOVE VAL,X     BYTE                    Store binary
C                     Z-ADD1         X
C                     MOVELHEXDIG    HEX2
C           HEX2      LOKUPDIG,X                       11  Lookup l.h  digit
C  N11                MOVE VAL,1     HEX2                    (If bad, use 0)
C                     TESTB'4'       VAL,X             11  Transfer
C   11                BITON'0'       BYTE                    binary
C                     TESTB'5'       VAL,X             11     value
C   11                BITON'1'       BYTE                       to
C                     TESTB'6'       VAL,X             11        left
C   11                BITON'2'       BYTE                         hand
C                     TESTB'7'       VAL,X             11           nybble
C   11                BITON'3'       BYTE
C                     END                                 End IF
C*
C                     ENDSR
C/EJECT
C*
C* BYTSHO routine   Convert binary to hex
C*
C*   Converts character in BYTE into hex digits in HEXDIG
C*
C*  Input- BYTE   Is the byte to be converted
C* Output- HEXDIG is a 2-character field containing the hex digits
C*
C           BYTSHO    BEGSR
C*
C                     MOVE BYTE      BITS                 Get binary byte
C                     BITOF'0123'    BITS                 Clear l.h  nybble
C                     Z-ADD1         X
C           BITS      LOKUPVAL,X                       11 Lookup r h  nybble
C                     MOVE DIG,X     HEXDIG               Store hex digit
C*
C                     MOVE BYTE      BITS                 Get binary byte
C                     BITOF'4567'    BITS                 Shift l.h  nybble
C                     TESTB'0'       BITS              11  into
C   11                BITON'4'       BITS                   r.h  nybble
C                     TESTB'1'       BITS              11
C   11                BITON'5'       BITS
C                     TESTB'2'       BITS              11
C   11                BITON'6'       BITS
C                     TESTB'3'       BITS              11
C   11                BITON'7'       BITS
C                     BITOF'0123'    BITS
C                     Z-ADD1         X
C           BITS      LOKUPVAL,X                       11 Lookup l.h  nybble
C                     MOVELDIG,X     HEXDIG               Store hex digit
C*
C                     ENDSR
C/EJECT
O*
O* Screen 0
O*
O@WORKSTNE                SCRN00
O                                   K8 'ATRSET00'
O                         MODNAM    8
O                         TYPE      9
O                         LIBNAM   24
O*
O* Screen 1
O*
O@WORKSTNE                SCRN01
O                                   K8 'ATRSET01'
O                         MASK1     8
O                         MASK2    16
O                         MASK3    24
O*
O* Screen 2
```

```
O*
O@WORKSTNE                SCRN02
O                                   K8  'ATRSET02'
O                         MASK4      8
O                         MASK5     10
O                         MASK6     18
O*
O* Screen 3
O*
O@WORKSTNE                SCRN03
O                                   K8  'ATRSET03'
O                         XMRT       2
O                         XREL       4
O                         XMOD1      6
O                         XMOD2      8
O                         XMOD3     10
O                         XDATE1    12
O                         XDATE2    14
O                         XDATE3    16
O                         XTIME1    18
O                         XTIME2    20
O*
O* Update directory records
O*
OMODCOPY E                UPDREC
O                         DIR,X      8
```

Figure 11-30

Screen format member ATRSETFM

```
*.   . . 1 ..  ... 2          3          4          5          6          7          8
SATRSET00            YY                                    G
D          23 128Y                                CLibrary Directory Entry
D          23 228Y                                C Attribute Set Utility
D        00610406Y                                CModule Name    .       X
D    .       .
DMODNAM  00080468Y  Y          01
D        00610506Y                                CModule Type (O,R,P,S)  X
D     .         .
DMODTYP  00010568Y  YA         01
D        00070570Y  Y          Y     Y
D        00610606Y                                CLibrary Name           X
D    .                             . .
DLIBNAM  00080668Y  Y          01
D          7824 3Y                                C                       X
D         Cmd7-Cancel                 Enter-Proceed
SATRSET01  0817      YY                                    G
DRECID      2 1 9Y  Y          Y     Y            C01
D        00150135Y                         Y      CAttribute Byte1
D        00280206Y                                CSSP attribute bit      X
D    .
DBIT1    00010235Y  Y Y   01      Y
DBIT2    00010237Y  Y Y           Y
DBIT3    00010239Y  Y Y           Y
DBIT4    00010241Y  Y Y           Y
DBIT5    00010243Y  Y Y           Y
DBIT6    00010245Y  Y Y           Y
DBIT7    00010247Y  Y Y           Y
DBIT8    00010249Y  Y Y           Y
D        00280251Y                                C   .     Module has oveX
Drlays
D        00320306Y                                CO-Privileged, P-Nolog..X
D       .!
D        00070339Y                                C!  ! ! !
D        00320347Y                                C!           PTF has beeX
Dn applied
D        00340406Y                                CNon-inquirable module..X
D    .      !
D        00030441Y                                C! !
D        00340445Y                                C!            .  NonbaseX
D SSP module
D        00360506Y                                CO-SFGR, P-PDATA/yes. ..X
D     .      !
D        00360543Y                                C!      .   . .     .  SoX
Durce required
D        00150635Y                         Y      CAttribute Byte2
```

```
0046 D         00280706Y                                            CDedicated module.. .   X
0047 D.
0048 DBIT1     00010735Y  Y Y           Y
0049 DBIT2     00010737Y  Y Y           Y
0050 DBIT3     00010739Y  Y Y           Y
0051 DBIT4     00010741Y  Y Y           Y
0052 DBIT5     00010743Y  Y Y           Y
0053 DBIT6     00010745Y  Y Y           Y
0054 DBIT7     00010747Y  Y Y           Y
0055 DBIT8     00010749Y  Y Y           Y
0056 D         00280751Y                                            C  .     Module has WTG X
0057 Dtable
0058 D         00320806Y                                            CNever Ending Program...X
0059 D..        !
0060 D         00070839Y                                            C! ! ! !
0061 D         00320847Y                                            C!                ProgramX
0062 D with UCS
0063 D         00340906Y                                            CModule has XREF fmt indX
0064 Dex.   ...  !
0065 D         00030941Y                                            C! !
0066 D         00340945Y                                            C!  .           .  ProgramX
0067 D has common
0068 D         00361006Y                                            CSecurity authority requX
0069 Dired. .     . !
0070 D         00361043Y                                            C!             .      CannoX
0071 Dt use // LOAD
0072 D         00151135Y                             Y              CAttribute Byte3
0073 D         00281206Y                                            C$WORK2 file required   X
0074 D.  .
0075 DBIT1     00011235Y  Y Y           Y
0076 DBIT2     00011237Y  Y Y           Y
0077 DBIT3     00011239Y  Y Y           Y
0078 DBIT4     00011241Y  Y Y           Y
0079 DBIT5     00011243Y  Y Y           Y
0080 DBIT6     00011245Y  Y Y           Y
0081 DBIT7     00011247Y  Y Y           Y
0082 DBIT8     00011249Y  Y Y           Y
0083 D         00281251Y                                            C       P-New copy of MRX
0084 DT req
0085 D         00321306Y                                            CTask is non-swappable..X
0086 D     .   .!
0087 D         00071339Y                                            C! ! ! !
0088 D         00321347Y                                            C!              Cross-refX
0089 Derencable
0090 D         00341406Y                                            CHigh-level dedication..X
0091 D.    .      !
0092 D         00031441Y                                            C! !
0093 D         00341445Y                                            C!              Must be trX
0094 Dansfered to
0095 D         00361506Y                                            CNeeds FORTRAN microcodeX
0096 D. . .        .!
0097 D         00361543Y                                            C!         .        ConfiguX
0098 Dration record
0099 D           7817 3Y                                            C                       X
0100 D           Cmd7-Cancel                           Enter-Next page
0101 SATRSET02  0817       YY                                                  BG
0102 DRECID    00020109Y  Y            Y       Y                    C02
0103 D         00150135Y                             Y              CAttribute Byte4
0104 D         00280206Y                                            CNeeds BASIC microcode..X
0105 D..   .
0106 DBIT1     00010235Y  Y Y           Y
0107 DBIT2     00010237Y  Y Y           Y
0108 DBIT3     00010239Y  Y Y           Y
0109 DBIT4     00010241Y  Y Y           Y
0110 DBIT5     00010243Y  Y Y           Y
0111 DBIT6     00010245Y  Y Y           Y
0112 DBIT7     00010247Y  Y Y           Y
0113 DBIT8     00010249Y  Y Y           Y
0114 D         00290251Y                                            C.     One copy executioX
0115 Dn only
0116 D         00320306Y                                            CPad module (spaceholderX
0117 D).. ..    !
0118 D         00070339Y                                            C! ! ! !
0119 D         00330347Y                                            C!          System TransiX
0120 Dent member
0121 D         00340406Y                                            CSUNGLOW program    .   X
```

```
0122 D.. ..      !
0123 D       00030441Y                                   C! !
0124 D       00350445Y                                   C!    ..   ...DDS load fX
0125 Dormat member
0126 D       00360506Y                                   CIBM supplied program...X
0127 D... .        !
0128 D       00370543Y                                   C!.   .  ....  Resides iX
0129 Dn a lib extent
0130 D       00150635Y                         Y         CAttribute Byte5
0131 D       00280706Y                                   C02-Data   14-DFU    18-X
0132 DPhone
0133 D       00010741Y  Y Y           Y
0134 D       00010743Y  Y Y           Y
0135 D       00290751Y                                   C33-COBOL  40-Unspec 58-X
0136 DQuery
0137 D         2808 6Y                                   C11-AutRsp 15-SFGR   19-X
0138 DSort
0139 D         290851Y                                   C34-FORTRN 53-EdText 59-X
0140 DCSP
0141 D         2809 6Y                                   C12-AutRpt 16-Menu   31-X
0142 DAsm
0143 D         290951Y                                   C35-RPG    54-FFText 5A-X
0144 DQryEnt
0145 D         2810 6Y                                   C13-BASICP 17-Mesg   32-X
0146 DBASIC
0147 D         291051Y                                   C36-WSU    55-HCText 5B-X
0148 DDocSrv
0149 D       00151135Y                         Y         CAttribute Byte6
0150 D       002812 6Y                                   CDynamically privileged.X
0151 D.   .
0152 DBIT1   00011235Y  Y Y           Y
0153 DBIT2   00011237Y  Y Y           Y
0154 DBIT3   00011239Y  Y Y           Y
0155 DBIT4   00011241Y  Y Y           Y
0156 DBIT5   00011243Y  Y Y           Y
0157 DBIT6   00011245Y  Y Y           Y
0158 DBIT7   00011247Y  Y Y           Y
0159 DBIT8   00011249Y  Y Y           Y
0160 D       00291251Y                                   C  .. ........ . .. ..  X
0161 D.. . .
0162 D       00321306Y                                   CDoes not need swap areaX
0163 D..... . !
0164 D       00071339Y                                   C! ! ! |
0165 D       00331347Y                                   C!..   ....... .........X
0166 D  . . .   .
0167 D       00341406Y                                   CEmulation member.......X
0168 D  .     . .!
0169 D       00031441Y                                   C! !
0170 D       00351445Y                                   C!. . .. ... ... .   ...X
0171 D           .   .
0172 D       00361506Y                                   CHas memory resident oveX
0173 Drlays.      .!
0174 D       00371543Y                                   C! ....    .....PC LAN miX
0175 Dcrocode member
0176 D         7717 4Y                                   CCmd2-Page back         X
0177 D       Cmd7-Cancel                   Enter-Next page
0178 SATRSET03  0817      YY                                     BG
0179 DRECID  00020109Y  Y           Y     Y              C03
0180 D       005202 6Y                         Y         CDescription            X
0181 D
0182 D       00120259Y                         Y         CValue
0183 D       005203 6Y                                   C0-MRTMAX count, P-x'FF'X
0184 D indicates MRT          .
0185 D       00020359Y  Y             Y
0186 D       005204 6Y                                   CRelease level..  .. . .X
0187 D     .      .. ..  ...      .. .
0188 D       00020459Y  Y             Y
0189 D       005205 6Y                                   CReference number.......X
0190 D            ..          .  .
0191 D       00020559Y  Y             Y
0192 D       00020562Y  Y             Y
0193 D       00020565Y  Y             Y
0194 D       005206 6Y                                   CDate member created/chaX
0195 Dnged...  .  .  .  . ... .. ..  .
0196 D       00020659Y  Y             Y
0197 D       00020662Y  Y             Y
```

```
0198 O          00020665Y  Y              Y
0199 D          005207 6Y                                     CTime member created/chaX
0200 Onged
0201 D          00020759Y  Y              Y
0202 D          00020762Y  Y              Y
0203 D            7717 4Y                                     CCmd2-Page back         X
0204 D          Cmd7-Cancel                  Enter-Update
```

Figure 11-31

First three library attribute bytes

```
                        Library Directory Entry
                         Attribute Set Utility

Module Name                                                   SPLARC
Module Type (O.R.P.S)                                         O
Library Name                                                  GARY

                             Attribute Byte1
SSP attribute bit            0 1 0 0 0 0 0 0      Module has overlays
O-Privileged. P-Nolog          ! ! ! ! ! !       PTF has been applied
Non-inquirable module            ! ! ! !           Nonbase SSP module
O-SFGR  P-PDATA/yes                ! !                Source required
                             Attribute Byte2
Dedicated module             0 0 0 0 0 0 0 0      Module has WTG table
Never Ending Program           ! ! ! ! ! !            Program with UCS
Module has XREF fmt index        ! ! ! !            Program has common
Security authority required        ! !             Cannot use // LOAD
                             Attribute Byte3
$WORK2 file required         0 0 0 0 0 0 0 0     P-New copy of MRT req
Task is non-swappable          ! ! ! ! ! !          Cross-referencable
High-level dedication            ! ! ! !          Must be transfered to
Needs FORTRAN microcode            ! !            Configuration record

                             Cmd7-Cancel              Enter-Next page
```

Figure 11-32

Second three library attribute bytes

```
                        Library Directory Entry
                         Attribute Set Utility

Module Name                                                   SPLARC
Module Type (O.R.P.S)                                         O
Library Name                                                  GARY

                             Attribute Byte4
Needs BASIC microcode        0 0 1 0 0 0 0 0      One copy execution only
Pad module (spaceholder)       ! ! ! ! ! !        System Transient member
SUNGLOW program                  ! ! ! !           DDS load format member
IBM supplied program               ! !            Resides in a lib extent
                             Attribute Byte5
02-Data    14-DFU    18-Phone      3 1        33-COBOL  40-Unspec 58-Query
11-AutRsp 15-SFGR    19-Sort                  34-FORTRN 53-EdText 59-CSP
12-AutRpt 16-Menu    31-Asm                   35-RPG    54-FFText 5A-QryEnt
13-BASICP 17-Mesg    32-BASIC                 36-WSU    55-HCText 5B-DocSrv
                             Attribute Byte6
Dynamically privileged       0 0 0 0 0 0 0 0
Does not need swap area        ! ! ! ! ! !
Emulation member                 ! ! ! !
Has memory resident overlays       ! !            PC LAN microcode member

Cmd2-Page back               Cmd7-Cancel              Enter-Next page
```

Figure 11-33
Miscellaneous library attributes

```
                      Library Directory Entry
                        Attribute Set Utility

  Module Name                                           SPLARC
  Module Type (O,R,P,S)                              .  O
  Library Name                                       .  GARY


  Description                                     Value
  O-MRTMAX count, P-x'FF' indicates MRT           00
  Release level                        .          51
  Reference number                   . .          00 00 33
  Date member created/changed                   . 88 08 25
  Time member created/changed                     13 10






Cmd2-Page back              Cmd7-Cancel               Enter-Update
```

Keeping Help Text in Source Members

by Mike Otey

Have you ever wanted to keep on-line help the way POP does — in easily maintainable source members — but you couldn't because the S/36 can't read library members without using custom assembler routines? An easy way to keep on-line help is available through POP's own tutorial facility.

By copying three load members — POPTUT, LIBR@TUT, and LIBR@F — from #POPLIB into your application library, you can easily create a procedure to display any source member with the built-in capability to scroll through the member using the Roll keys. As the example in Figure 11-34 illustrates, you simply supply, starting in position 1 of the LDA, the source member name to be displayed and then execute POPTUT. The source member (in Figure 11-34, DOC0001) is displayed. Using this technique, you can create and maintain on-line help text using FSEDIT, and you won't have to recompile screen formats to implement text changes.

Figure 11-34
Using POPTUT to create on-line help text

```
// LOCAL OFFSET-1,DATA-'DOC0001'
// LOAD POPTUT
// RUN
```

Unlocking a BASIC Source Program

by Mark E. Bonney

Code on diskette:
Procedure BASUNL

To enforce source code security, S/36 BASIC offers you the option to LOCK library members before saving them to a library. The LOCK feature lets you change the source program by line number, but does not let you view a program listing. To list the program for major revisions, IBM suggests you make a copy of the program before LOCKing the production version of the program. IBM left out the ability to "unlock" a source member after the fact, but you can use the OCL in Figure 11-35 to unlock a BASIC source program (which BASIC stores in R-type library members) using IBM's $FEFIX program patch utility.

If you run this procedure on a BASIC module that has never been locked, SSP issues message SYS-3330, "Check byte in DATA statement incorrect or missing." Taking option 2 to this message ends the procedure.

Figure 11-35

Procedure BASUNL to unlock a BASIC source program

```
// TAG ENTRY
// IF ?1?/ * 'ENTER THE BASIC MODULE NAME TO UNLOCK'
// IF ?1R?=END CANCEL
// TAG ENTRY2
// IF ?2?/ * 'ENTER THE LIBRARY NAME OF THE BASIC MODULE TO UNLOCK'
// IF ?2R?=END CANCEL
// IF DATAF1-?2'?SLIB?'? GOTO ENTRY3
// * 'THE LIBRARY YOU REQUESTED DOES NOT EXIST'
// * ''
// GOTO ENTRY2    ?2F''?
// TAG ENTRY3
// IF SUBR-'?1?,?2?'   GOTO ENTRY4
// * 'THE BASIC MODULE YOU REQUESTED DOES NOT EXIST IN ?2?'
// * ''
// GOTO ENTRY1    ?1F''?  ?2F''?
// TAG ENTRY4
// * ''
// * 'UNLOCKING BASIC MODULE'
// LOAD $FEFIX
// RUN
HDR
PTG  R?1?,,?2?
DATA 00,000C,01
END
```

Adding Members to and Compressing #LIBRARY

answered by Mel Beckman

Code on diskette:
Procedure LIB#DECR
Message member LIB#2518

Q Is there a way to automatically increase #LIBRARY to add user members and then to decrease it to an optimum size in a procedure?

A To increase #LIBRARY, you must first run a COMPRESS FREELOW to make space available immediately following #LIBRARY on your disk. Then you can use the ALOCLIBR procedure to increase #LIBRARY's size to accommodate your new members.

To decrease #LIBRARY to an optimum size, try the following solution:

a. Create a source member: LIB#2518

```
SYS
2518 2,1               CANNOT REDUCE THIS LIBRARY TO GIVEN
SIZE
```

b. Create a procedure: LIB#DECR

```
RESPONSE LIB#2518           SET AUTO-RESPONSE
NOHALT 1,JOB
// EVALUATE P1=5100         STARTING MINIMUM SIZE
*
// TAG LOOP
// EVALUATE  P1=?1?+100     KEEP INCREASING UNTIL SUCCESS
*
ALOCLIBR #LIBRARY,?1?
// IF ?CD?=3721  GOTO LOOP
```

Of course, you can give the initial size and loop increment the value that suits your system best.

Resizing #LIBRARY

answered by Ron Mendel

Q I recently attempted to add an IBM program product to my S/36 system configuration, only to be informed by SSP that not enough disk space was available to store the new software. I know there is room on the disk, but when I attempt to increase the size of #LIBRARY to store the program products, I cannot get the system to accept my change. What's my problem and how do I solve it?

A On the S/36, SSP allocates space for #RPGLIB, the system security file, and other system files immediately following #LIBRARY on the low-address end of the disk. These system files occupy the physical locations on disk that SSP otherwise would allocate to #LIBRARY when you attempt to resize it. (SSP allocates space to files and libraries in contiguous blocks.) Therefore, any direct attempt to increase the size of #LIBRARY, the Task Work Area, or the system history file will meet with the difficulty you describe.

Before you resize #LIBRARY, you need to move other files and libraries away from #LIBRARY, thereby freeing contiguous disk space so that you can increase your #LIBRARY allocation. On the S/36, this space can be freed with the COMPRESS procedure, specifying A1 as the first parameter and LOW (or FREELOW) as the second parameter, or by entering:

```
// LOAD $FREE
// RUN
// COMPRESS DISK-A1,FREE-LOW
// END
```

The $FREE utility will move all disk objects except #LIBRARY to the high-address end of disk A1 and accumulate free space at the low-address end of the disk, thereby creating room for you to expand the size of #LIBRARY.

Removing PTF Libraries

answered by Mike Patton and Ed Girou

Q Several libraries were created during the installation of some PTFs on our S/36. Are these libraries useful by-products of the PTF installation, or can they be deleted?

A Those libraries are PTF backup libraries, which serve no useful purpose *unless* you have a rogue PTF that you must remove. The PTF REMOVE procedure relies on the backup libraries to reverse the effect of a PTF application for any changed modules. Although you may never need these libraries, it's a good idea to SAVELIBR them before you delete them from your system. Then all you need do is restore them to remove one or more bad PTFs. If you need to remove a PTF and you delete the libraries without doing a SAVELIBR, however, you will have to reload the system library, reapply the PTFs, and then remove the ones causing problems.

MAPICS

CHAPTER

12

Reducing Time and Diskettes for MAPICS SAVE

by B. Booth Deakins

Diskette compression was an enhancement provided with Release 3.0 of the SSP on the S/36. However, during the file save function, MAPICS I and II do not take advantage of the diskette compress feature. You can add this feature to your SAVE procedure by changing a single line in the MAPICS procedure AMZPKC. You need to add the COMPRESS-YES parameter to line 42 (approximately) of the AMZPKC procedure so that it reads as it does in Figure 12-1. You will see a significant reduction in both the save time and number of diskettes used.

Figure 12-1

Modification to MAPICS AMZPKC procedure

```
// IF ?L'127,1'?/M          COPYALL TO-I1,GROUP-M,COMPRESS-YES
```

Deleting MAPICS Backup Diskettes

by Ray Trimber

Code on diskette:
Procedure DELMAP

MAPICS file backups require that no one be signed on to the MAPICS library, AMALIB. This restriction either cuts into valuable user time if backups are performed during normal working hours or forces an operator or manager to work past quitting time if backup is postponed until after hours.

One way to reduce the time involved is to choose not to delete old backup files from diskettes during your backup routine. Instead, at your convenience, use the procedure in Figure 12-2 to delete the old backup files. This procedure can be run at any time and does not interfere with other users on the system. Note that the procedure in Figure 12-2 bypasses MAPICS security, so you may want to build in your own security measures.

Figure 12-2

Procedure DELMAP

```
// * 'Diskette magazine delete'
// * 'Do you want to delete M2? (Y/N)'
// IF ?1R?/ IF ?1'N'?/
// LOAD $DELET
// RUN
// REMOVE UNIT-I1,LABEL-ALL,PACK-AMBACK,LOCATION-M1.01,ENDLOC-M1.10
// IF ?1?/Y REMOVE UNIT-I1,LABEL-ALL,PACK-AMBACK,LOCATION-M2.01,ENDLOC-M2.10
// END
```

Reorganizing MAPICS Files That Use Alternate Indexes

by Perry Gardai

program by Dale S. Walker

Code on diskette:
Procedure AIUTIL
RPG programs AIBLD, AIDSP, AIDEL
Screen format member AIDSPFM

It's 2 a.m., and you are tossing and turning. You left the office last night after starting a MAPICS reorg on your S/36, and you are hoping there will be no unplanned system halts requiring a MAPICS restore in the morning. But you keep having problems with the alternate indexes on your MAPICS parent files, and you are dreading any "surprise" that may be waiting at the office.

You have probably found the Alternate Index (AI) functions valuable as an additional index to a data file that allows you to access a file by a different key. But AIs can affect deletions and reorganizations of their parent files, especially when you have user-defined AI files attached to MAPICS master files. The MAPICS procedure AXZPZ8 does a good job of reorganizing the MAPICS parent files to free disk space occupied by deleted or inactive records; however, it does not recognize the existence of AI files attached to the MAPICS parent file. If, during a MAPICS reorg, procedure AXZPZ8 tries to delete a MAPICS master file that has an AI file attached to it, the system will issue the system message SYS-1627, "Cannot Delete Physical File." The only recovery options provided are 2 (cancel and continue) and 3 (cancel). With either response, you probably will be forced into a master file restore, a time-consuming chore. The AIUTIL utility solves this problem, and lets you sleep better, by selectively deleting your AI files before the reorg commences.

AIUTIL is a utility composed of a series of system procedures and application programs that identify and selectively delete up to 10 AI files for each MAPICS parent file (few, if any, MAPICS parent files would have more than 10 AI files). Procedure AIUTIL (Figure 12-3) is called from MAPICS procedure AXZPZ8 before the actual reorganization routines are executed and uses three programs to process AI files, allowing the reorg to progress unhampered.

Program AIBLD (Figure 12-4) reads a disk file that contains a disk Volume Table of Contents (VTOC) sequenced by name and creates the indexed file AIOUT?WS?, which contains the parent file name as the key, and up to 10 associated alternate index file names.

Program AIDSP (Figure 12-5) displays each record from the file created by program AIBLD and gives you the option to delete the AI files associated with the MAPICS parent file currently displayed. Program AIDEL (Figure 12-6) passes the records selected for deletion to the calling procedure via the LDA, at which point the records are deleted.

The process begins when MAPICS procedure AXZPZ8 calls procedure AXZPZ7, which renames the MAPICS control file from M.SYSCTL to M.SYSXXX. This name change puts the MAPICS application into a dedicated mode in preparation for the upcoming reorganization. Procedure AXZPZ8 then calls procedure AIUTIL.To accomplish this call, you should modify procedure AXZPZ8 with a single line of code immediately after the call to procedure AXZPZ7 (Figure 12-7).

Procedure AIUTIL (Figure 12-3) first displays a formatted message that informs you the utility will search for all AI files attached to the MAPICS parent files. To allow AIUTIL to be executed more efficiently, the procedure sets the region size to 64 K via the // REGION statement.

The next two sections of procedure AIUTIL, the $LABEL and the $UASF routines, create a disk file (CAT?WS?) of a catalog listing that will be read by program AIBLD. The $LABEL routine creates a catalog listing with the forms ID of CTLG. The PRIORITY-0 parameter on the // PRINTER statement puts the listing on hold in the spool file. Then $UASF copies the catalog listing into disk file CAT?WS?. The RELCANS-CANCEL parameter on the COPYPRT statement removes spool entry CTLG from the spool file. The RETAIN-J parameter will remove disk file CAT?WS? from disk when the procedure terminates. The procedure then checks for the existence of data file AIOUT?WS?. If file AIOUT?WS? already exists on disk, the $DELET routine deletes it. If it does not exist, the procedure branches to the // TAG RUNBLD statement and program AIBLD is executed.

Program AIBLD (Figure 12-4) reads catalog file CAT?WS? and outputs into file AIOUT?WS? one record for each MAPICS parent file name it finds. Each record contains the MAPICS parent file name as the key (positions 2 through 9) and a data portion composed of a 10-element array. Each element of the array can contain the name of one AI file attached to a specific MAPICS parent file.

The I-specs for the catalog file (in this example, file CATIN) ensure only records from the catalog file that identify AI files will be used to build the new file. The data contained in file CATIN records includes the AI file name, defined as data field AIFILE, and the name of the parent file, defined as field SYSKEY. Within the MAPICS control file, only the last six characters of the file name are logged; therefore, the data field SYSKEY does not use the M. — the first two characters. Program AIBLD uses field SYSKEY to chain to MAPICS control file SYSCTL. If the chain fails (i.e., the file being processed is not a MAPICS master), indicator 90 is set on, the remainder of the C-specs are bypassed, and no records are added to file AIOUT?WS?

If the chain is successful, then the parent file name just processed is indeed a MAPICS parent file, and the entire parent file name (all eight characters), now defined as data field PARENT, is used to chain to the AIOUT?WS? file. If the chain fails, the program sets on indicator 91, which means program AIBLD is processing this particular MAPICS parent file and an associated AI file for the first time. Therefore, a record will be added to file AIOUT?WS? with the MAPICS parent file name as the key and the AI file name as the contents of the first element of array ARR.

As each subsequent record is processed for this particular MAPICS parent file, the chain to file AIOUT?WS? will be successful (indicator 91 will be off). Under this condition, the program then will do a lookup to find the first blank element of array ARR. Once this is accomplished, the blank element is loaded with the AI name currently being processed and file AIOUT?WS? is updated with the current contents of array ARR. This process continues until the entire catalog is processed and control returns to procedure AIUTIL. If, at this point, file AIOUT?WS? does not contain any records, indicating that none of the MAPICS parent files currently have AI files attached, the procedure branches to TAG ENDAI, file AIOUT?WS? is deleted, and the procedure terminates.

If file AIOUT?WS? does contain records, the procedure loads the LDA with the user ID and the workstation ID in preparation for the execution of program AIDSP (Figure 12-5). Program AIDSP displays each MAPICS master file and its associated AI files. You are given three processing options — Y, N, or Command key 24 — to indicate which action should be performed on each record of file AIOUT?WS?.

Program AIDSP begins by displaying one screen for each MAPICS parent file and all of its AI files (Figure 12-8a is Display Screen AI01; Figure 12-8b is the screen format member). If you enter Y into the OPTION field, the program sets on indicator 21 and writes character D to the status byte (position 1) of the AIOUT?WS? record. The presence of the character D in the first position indicates that all AI files associated with this particular MAPICS master file are to be deleted. If you enter N into field OPTION, the program sets on indicator 20, and the status byte is not updated with a D. After program AIDSP has processed each record in file AIOUT?WS?, the program terminates, and control returns once again to procedure AIUTIL. If at any point during the execution of program AIDSP you press Command key 24 to cancel the selection program, the program writes the character C to position 424 of the LDA and sets on indicator LR. If a C is in position 424 of the LDA, the procedure branches to TAG ENDAI, deletes the file AIOUT?WS?, and terminates.

The final sections of procedure AIUTIL are responsible for the actual deletion of the selected AI files and for printing an audit report. The deletion is completed by using program AIDEL (Figure 12-6), which processes one record from file AIOUT?WS?. If the record is marked for deletion (the D in position 1), the program loads the LDA with the names of the AI files attached to that record and prints the corresponding audit report. Program AIDEL then terminates, and control returns to procedure AIUTIL.

The $DELET routine checks the appropriate positions of the LDA and deletes the file name stored there. After the $DELET terminates, the procedure loops back up to program AIDEL and repeats the cycle until all the records from file AIOUT?WS? are processed.

The last few executable lines of the procedure that follow the // TAG ENDAI statement delete file AIOUT?WS?. Then the procedure ends, and control returns to the MAPICS master procedure AXZPZ8, at which time it can continue and reorganize the files.

There are a few aspects to this procedure that may not be obvious and may require you to make extensive modifications. First, this procedure processes only original MAPICS parent files (M. files) that are logged in the MAPICS control file M.SYSCTL. Therefore, AIUTIL lets you delete only files that have a parent M.file and a SYSCTL record. Second, AIUTIL will accommodate only 10 AI files attached to any one MAPICS parent file. If you have more than 10 AI files attached to any one MAPICS parent file, you will have to modify this procedure. However, its logic and basic structure can be maintained. Finally, and perhaps most important, AIUTIL does not rebuild the deleted AI files. This situation could cause some serious problems if your existing applications do not check for the presence of required AI files and rebuild them before executing each application program that makes use of them. If you have AI files that you use often, you may want to modify the utility to rebuild the AI files after a reorg. Otherwise, the procedure that uses the AI files should test for their existence and rebuild them when necessary.

These limitations aside, AIUTIL has proved to be an invaluable tool in my MAPICS shop. I no longer lose sleep worrying about the results of the unattended MAPICS reorgs that are run every night. I know there won't be any unplanned system halts requiring a MAPICS restore waiting for me in the morning.

Figure 12-3

Procedure AIUTIL

```
*  PROCEDURE NAME  AIUTIL    (ALTERNATE INDEX CHECK UTILITY)
*  DATE COMPLETED  08/86     DALE S. WALKER
*  CALLING PROCEDURE: AXZPZ8 (MAPICS FILE STATUS / REORGANIZE PROCEDURE)
*  FUNCTION: SEE END OF PROCEDURE FOR FURTHER DOCUMENTATION
*
*
*
// * ' *****************************************************'
// * ' * NOW SEARCHING FOR ALTERNATE INDEX FILES APPENDED  *'
// * ' *           TO YOUR MAPICS MASTER FILES            *'
// * ' *                 PLEASE STAND BY                  *'
// * ' *****************************************************'
// * ' '
// * ' '
*
*      RUN A CATALOG BY NAME AND HOLD THE SPOOL FILE ENTRY
*
//          REGION SIZE-64
//          LOAD $LABEL
//          PRINTER NAME-$SYSLIST,FORMSNO-CTLG,PRIORITY-0
//          RUN
//          DISPLAY LABEL-ALL,UNIT-F1
//          END
//          REGION SIZE-24
*
*      COPY THE SPOOL FILE ENTRY TO DISK
*
//          LOAD $UASF
//          RUN
```

```
//           SPOOL SPOOLID-FCTLG,NAME-CAT?WS?,RELCANS-CANCEL,RETAIN-J
//           END
*
*       BUILD A FILE THAT CONSISTS OF THE PARENT M.FILE AS THE KEY,
*       AND INCLUDE ALL OF ITS ASSOCIATED AI FILES
*
//           IFF DATAF1-AIOUT?WS?     GOTO RUNBLD
//           LOAD $DELET
//           RUN
//           SCRATCH LABEL-AIOUT?WS?,UNIT-F1
//           END
*
//          TAG RUNBLD
//           LOAD AIBLD
//           FILE NAME-CATIN,LABEL-CAT?WS?,DBLOCK-40
//           FILE NAME-AIOUT,LABEL-AIOUT?WS?,DISP-NEW,RECORDS-25,EXTEND-15
// IF DATAF1-M.SYSCTL           FILE NAME-SYSCTL,LABEL-M.SYSCTL,DISP-SHRRM
// ELSE                         FILE NAME-SYSCTL,LABEL-M.SYSXXX,DISP-SHRRM
//           RUN
*
*       CANCEL ONLY IF THERE WERE NO AI FILES ATTACHED TO AN M.FILE
*
//           IF ?F'A,AIOUT?WS?'?/0    GOTO ENDAI
*
*       DISPLAY THE PARENT FILE ALONG WITH UP TO 10 OF ITS ASSOCIATED
*       AI FILES, AND ALLOW FOR AN OPTION TO DELETE THE AI FILES
*
//           LOCAL OFFSET-414,DATA-'?USER?'
//           LOCAL OFFSET-422,DATA-'?WS? '
//           LOAD AIDSP
//           FILE NAME-AIOUT,LABEL-AIOUT?WS?
//           RUN
*
*       IF CK24 WAS ENTERED IN PGM-DSPAI, DO NOT DELETE ANY FILES.
*
//           IF ?L'424,1'?/C     GOTO ENDAI
*
*       PRINT AN AUDIT REPORT LISTING THE AI FILES THAT WERE SELECTED
*       FOR DELETION, AND PLUG THE LDA WITH THE FILE NAMES
*
//          TAG DELET
//           LOCAL OFFSET-424,DATA-' ',BLANK-89
//           LOAD AIDEL
//           FILE NAME-AIOUT,LABEL-AIOUT?WS?
//           PRINTER NAME-PRINTER,CONTINUE-YES
//           RUN
*
*        GO TO ENDAI IF THERE ARE NO MORE FILES TO DELETE
*
//           IF ?L'424,1'?/C     GOTO ENDAI
*
*       DELETE THE SELECTED AI FILES AND LOOP TO RETRIEVE NEXT FILES
*       TO BE DELETED
*
//           * 'NOW DELETING AI FILES FOR PARENT - ?L'425,8'?'
//           LOAD $DELET
//           RUN
//           IFF ?L'433,8'?/     SCRATCH LABEL-?L'433,8'?,UNIT-F1
//           IFF ?L'441,8'?/     SCRATCH LABEL-?L'441,8'?,UNIT-F1
//           IFF ?L'449,8'?/     SCRATCH LABEL-?L'449,8'?,UNIT-F1
//           IFF ?L'457,8'?/     SCRATCH LABEL-?L'457,8'?,UNIT-F1
//           IFF ?L'465,8'?/     SCRATCH LABEL-?L'465,8'?,UNIT-F1
//           IFF ?L'473,8'?/     SCRATCH LABEL-?L'473,8'?,UNIT-F1
//           IFF ?L'481,8'?/     SCRATCH LABEL-?L'481,8'?,UNIT-F1
//           IFF ?L'489,8'?/     SCRATCH LABEL-?L'489,8'?,UNIT-F1
//           IFF ?L'497,8'?/     SCRATCH LABEL-?L'497,8'?,UNIT-F1
//           IFF ?L'505,8'?/     SCRATCH LABEL-?L'505,8'?,UNIT-F1
//           END
//           GOTO DELET
*
*       DELETE THE AIOUT FILE AND RETURN
*
//          TAG ENDAI
//           LOAD $DELET
//           RUN
//           SCRATCH LABEL-AIOUT?WS?,UNIT-F1
```

```
//            END
//            RETURN
*
*
*
*  FUNCTION. THIS PROCEDURE ALLOWS YOU TO PREVENT THE SYSTEM FROM ISSUING
*            YOU THE ERROR MESSAGE, 'SYS-1627  CANNOT DELETE PHYSICAL
*            FILE (FILENAME)',  WHILE EXECUTING A MAPICS MASTER FILE
*            REORGANIZATION. THIS ERROR CONDITION IS THE RESULT OF TRYING
*            TO DELETE A FILE FROM DISK WHEN IT HAS ALTERNATE INDEX
*            (AI) FILES APPENDED TO IT
*            WHILE RUNNING THIS PROCEDURE, THE OPERATOR HAS THE OPTION
*            TO DELETE AI FILES FOR ANY SELECTED MAPICS PARENT FILE,
*            BEFORE CONTINUING WITH THE MAPICS MASTER FILE REORGANIZATION.
*
*  LDA USAGE·        FROM TO    USAGE
*                    414  421   USER ID
*                    422  423   WORKSTATION ID
*                    424  424   CANCEL/EOJ FLAG
*                    425  432   PARENT FILE NAME
*                    433  512   AI FILE NAMES
* END OF AIUTIL
```

Figure 12-4

Program AIBLD

```
*. . . . 1 . . .  . . 2  .     3  .    . 4  .  . 5     . 6     . 7       8
     H                                                                 AIBLD
     F*  PROGRAM NAME. AIBLD      (BUILD THE AI FILE)
     F*  DATE COMPLETED:  8/86    DALE S. WALKER
     F*  CALLING PROCEDURE: AIFILE
     F*  FUNCTION· THIS PROGRAM READS A DISK FILE THAT CONTAINS A
     F*            DISK VTOC (SEQUENCED BY NAME), AND CREATES AN
     F*            INDEXED FILE THAT CONTAINS THE PARENT FILE NAME AS
     F*            THE KEY, AND UP TO 10 OF ITS ASSOCIATED ALTERNATE
     F*            INDEX FILE NAMES
     F*
     F*      NOTE  ONLY PARENT FILES THAT ARE MAPICS MASTER FILES
     F*            WILL BE PROCESSED  ALL USER CREATED FILES THAT
     F*            BELONG TO THE 'M ' FILE GROUP WILL NOT BE PROCESSED.
     F/SPACE 2
     F*************************************************************
     F*               INDICATOR USAGE AND DEFINITIONS             *
     F*************************************************************
     F*  01 - THIS AI FILE HAS AN M.FILE FOR A PARENT              *
     F*  30 - SUCCESSFUL ARRAY LOOKUP FOR AN UNUSED ELEMENT        *
     F*  90 - CHAIN ERROR INDICATOR FOR FILE - SYSCTL              *
     F*  91 - CHAIN ERROR INDICATOR FOR FILE - AIOUT               *
     F*************************************************************
     F/SPACE 2
     FCATIN   IP     150 150            DISK
     FAIOUT   UC     128 128R 8AI     2 DISK                      A
     FSYSCTL  IC     128 128R 6AI     3 DISK
     E*  THIS ARRAY WILL HOLD UP TO 10 ALTERNATE INDEX FILE NAMES
     E/SPACE
     E                     ARR        10  8
     I*  CATIN INPUT FILE SPECIFICATIONS
     I/SPACE
     ICATIN   NS  01 106 CM 107 C
     I                                      11  18 AIFILE
     I                                     106 113 PARENT
     I                                     108 113 SYSKEY
     I        NS
     I/SPACE 2
     I*  AIOUT INPUT FILE SPECIFICATIONS
     I/SPACE
     IAIOUT   NS
     I                                       2   9 PARNT
     I                                      10  89 ARR
     I/SPACE 2
     I*  SYSCTL INPUT FILE SPECIFICATIONS
     I/SPACE
     ISYSCTL  NS       1 CC    2 CD
```

```
I                                      1   2 RCDCD
I                                      3   8 SCKEY
I/SPACE
I        NS
C        N01          GOTO BYPASS                        NOT AN M.FILE FOR A PARENT - BYPASS
C*
C           SYSKEY    CHAINSYSCTL              90        CHECK IF PARENT IS A MAPICS FILE
C   90                SETOF                    01        NO HIT - BYPASS THIS RECORD
C   90                GOTO BYPASS
C*
C           PARENT    CHAINAIOUT               91        GET PARENT'S 'AIOUT' RECORD
C   91                GOTO BYPASS                        NO HIT - ADD A RECORD
C                     Z-ADD1         X        20         INITIALIZE ARRAY SUBSCRIPT
C           *BLANKS   LOKUPARR,X                     30  GET AN UNUSED ARRAY ELEMENT
C   30                MOVE AIFILE    ARR,X               PLUG THE ELEMENT W/THE FILE NAME
C           BYPASS    TAG
O*  AIOUT FILE ADDITIONS
O/SPACE
OAIOUT   DADD      01 91
O                         PARENT     9
O                         AIFILE    17
O/SPACE 2
O*  AIOUT FILE UPDATES
O/SPACE
O        D         01N91
O                         ARR       89
```

Figure 12-5

Program AIDSP

```
*... . . 1    .  .. 2   . ... 3  ..  .  4 ..   ... 5 ... ... 6 ..   .. 7 ... ... 8
H                                                                            AIDSP
F*  PROGRAM NAME. AIDSP      (DISPLAY AI FILES)
F*  DATE COMPLETED:  8/86     DALE S. WALKER
F*  CALLING PROCEDURE: AIUTIL
F*  FUNCTION· THIS PROGRAM DISPLAYS THE PARENT FILE ALONG
F*            WITH UP TO 10 OF ITS ASSOCIATED AI FILES. THE
F*            OPERATOR THEN HAS THE OPTION TO SELECT THE AI
F*            FILES FOR DELETION
F/SPACE 2
F*************************************************************
F*                INDICATOR USAGE AND DEFINITIONS            *
F*************************************************************
F*  KY - OPERATOR HAS ENTERED CK24 TO CANCEL                 *
F*  LR - LAST RECORD INDICATOR                               *
F*  01 - ACTIVE PRIMARY INPUT RECORD                         *
F*  20 - OPERATOR CHOSE NOT TO DELETE THE AI FILE(S)         *
F*  21 - OPERATOR CHOSE TO DELETE THE AI FILE(S)             *
F*  90 - SFGR ERROR INDICATOR - INVALID OPTION WAS ENTERED   *
F*************************************************************
F/SPACE 2
FAIOUT   UP    128 128           DISK
FTUBE    CD        128           WORKSTN
I*  AIOUT INPUT FILE SPECIFICATIONS
I/SPACE
IAIOUT   NS  01    1 C
I                                      1   1 ACREC
I                                      2   9 PARENT
I                                     10  89 ARR
I        NS
I/SPACE 2
I*  TUBE INPUT FILE SPECIFICATIONS
I/SPACE
ITUBE    NS
I                                      1   1 OPTION
I/SPACE 3
I*  LOCAL DATA AREA INPUT SPECIFICATIONS
I/SPACE
I           UDS
I                                    414 421 USER
I                                    422 423 WSID
I                                    424 424 CANCL
C        N01          GOTO BYPASS                        PRIMARY RECORD IS NOT ACTIVE
```

```
C                     EXCPTWRITE                          ISSUE SCRN FMT - AI01
C          RDTUBE     TAG
C   90                SETOF                     90        CLEAR SFGR ERROR INDICATOR
C                     READ TUBE                           READ THE WORKSTATION FILE
C   KY                SETON                     LR        CK24 - EOJ
C   KY                MOVE 'C'       CANCL
C   KY                GOTO BYPASS
C*
C          OPTION     COMP 'N'                      20    N - DO NOT DELETE AI FILES
C          OPTION     COMP 'Y'                      21    Y - DELETE AI FILES
C     N20N21          SETON                     90        ERROR CONDITION - INVALID OPTION
C   90                EXCPTWRITE                          ISSUE ERROR MSG
C   90                GOTO RDTUBE                         GO AND RE-READ THE WORKSTATION FILE
C*
C          BYPASS     TAG
O*  WORKSTATION FILE OUTPUT SPECIFICATIONS
O/SPACE
O*  RELEASE THE WORKSTATION ON CK24
O/SPACE
OTUBE    DR        KY
O/SPACE 2
O*  SCREEN FORMAT - AI01
O/SPACE
O        E                WRITE
O                                  K4 'AI01'
O                         UDATE Y    8
O                         WSID      10
O                         PARENT    18
O                         ARR       98
O                  90              123 'INVALID OPTION-TRY AGAIN'
O/SPACE 2
O*  AIOUT FILE OUTPUT SPECIFICATIONS
O*  (FLAG THE PARENT TO HAVE ITS AI FILES DELETED)
O/SPACE
OAIOUT   D         01 21NKY
O                                    1 'D'
```

Figure 12-6

Program AIDEL

```
*.   .. 1 . . .. 2      . . 3    . . . 4   .  .. 5 ...    . 6 . . .    7         8
    H                                                  1                      AIDEL
    F*  PROGRAM NAME: AIDEL       (AI FILE DELETION SELECTION)
    F*  DATE COMPLETED:  8/86     DALE S. WALKER
    F*  CALLING PROCEDURE: AIUTIL
    F*  FUNCTION. THIS PROGRAM READS THE AIOUT RECORDS THAT HAVE HAD
    F*            THEIR AI FILES SELECTED FOR DELETION BY THE PROGRAM
    F*            AIDSP. THE AI FILE NAMES ARE PASSED TO THE CALLING
    F*            PROCEDURE VIA THE LDA AND DELETED  AN AUDIT REPORT
    F*            IS PRINTED THAT CONTAINS A LISTING OF AI FILES THAT
    F*            ARE TO BE DELETED.
    F/SPACE 2
    F*************************************************************
    F*               INDICATOR USAGE AND DEFINITIONS             *
    F*************************************************************
    F*  LR - LAST RECORD INDICATOR                               *
    F*  OF - PRINTER FILE OVERFLOW INDICATOR                     *
    F*  01 - AIOUT PRIMARY INPUT RECORD - AI FILES ARE SEL  FOR DEL.*
    F*  1P - PRINTER FILE FIRST PAGE INDICATOR                   *
    F*  20 - INDICATES                                           *
    F*  21 -   THAT                                              *
    F*  22 -     THE                                             *
    F*  23 -       ARRAY                                         *
    F*  24 -         ELEMENT                                     *
    F*  25 -           CONTAINS                                  *
    F*  26 -             AN                                      *
    F*  27 -               AI                                    *
    F*  28 -                 FILE                                *
    F*  29 -                   NAME                              *
    F*  30 - LR WAS SETON MANUALLY   (N30 RPG SETON LR)          *
    F*************************************************************
    F/SPACE 2
    FAIOUT   UP    128 128            DISK
```

```
FPRINTER O     132 132     OF      PRINTER
E*  THIS ARRAY WILL HOLD UP TO 10 ALTERNATE INDEX FILE NAMES
E/SPACE
E                    ARR        10  8
I*  AIOUT INPUT SPECIFICATIONS
I/SPACE
IAIOUT   NS  01    1 CD
I                                         1   1 ACREC
I                                         2   9 PARENT
I                                        10  89 ARR
I        NS
I/SPACE 2
I*  LOCAL DATA AREA INPUT SPECIFICATIONS
I/SPACE
I          UDS
I                                       414 421 USERID
I                                       422 423 WSID
I                                       424 424 CANCL
I                                       425 432 PARENT
I                                       433 512 ARR
C                     TIME           TME     60         CAPTURE THE SYS. TIME FOR HEADINGS
C        N01          GOTO BYPASS                       AI FILES NOT SELECTED FOR DELETION
C           ARR,1     COMP *BLANKS               20     CHECK IF THIS ELEMENT CONTAINS AN
C           ARR,2     COMP *BLANKS               21       AI FILE NAME (20-29)
C           ARR,3     COMP *BLANKS               22
C           ARR,4     COMP *BLANKS               23
C           ARR,5     COMP *BLANKS               24
C           ARR,6     COMP *BLANKS               25
C           ARR,7     COMP *BLANKS               26
C           ARR,8     COMP *BLANKS               27
C           ARR,9     COMP *BLANKS               28
C           ARR,10    COMP *BLANKS               29
C                     SETON                      LR30   SETON LR TO DELETE THESE FILES
C           BYPASS    TAG
CLRN30                MOVE 'C'       CANCL              EOF
O*  AIOUT FILE UPDATES
O/SPACE
O*  FLAG THIS RECORD AS ALREADY BEING PROCESSED
O/SPACE
OAIOUT   D        01
O                                    1 ' '
O/SPACE 3
O*  FILE DELETION AUDIT REPORT
O/SPACE
OPRINTER D  2     01
O                                   30 'ALTERNATE INDEX FILE'
O                                   44 'CHECK UTILITY'
O                                   54 'PAGE'
O                         PAGE  Z   59
O/SPACE
O        D  3     01
O                                    8 'DATE'
O                         UDATE Y   17
O                                   24 'TIME'
O                         TME       33 '         '
O                                   40 'USER'
O                         USERID    49
O/SPACE
O        D  2     01
O                                   11 'PARENT FILE'
O                         PARENT    20
O                                   42 'HAS HAD THE FOLLOWING'
O                                   59 'AI FILES DELETED'
O/SPACE
O        D  1     01 20
O                         ARR,1     25
O/SPACE
O        D  1     01 21
O                         ARR,2     25
O/SPACE
O        D  1     01 22
O                         ARR,3     25
O/SPACE
O        D  1     01 23
O                         ARR,4     25
O/SPACE
```

```
0         D  1      01 24
0                           ARR,5      25
0/SPACE
0         D  1      01 25
0                           ARR,6      25
0/SPACE
0         D  1      01 26
0                           ARR,7      25
0/SPACE
0         D  1      01 27
0                           ARR,8      25
0/SPACE
0         D  1      01 28
0                           ARR,9      25
0/SPACE
0         D  1      01 29
0                           ARR,10     25
```

Figure 12-7

Modification to MAPICS procedure AXZPZ8

```
*********************************************************************
*                      AXZPZ8 HAS BEEN MODIFIED                     *
*********************************************************************
*                                                                   *
*  MODIFIED TO EXECUTE THE PROC - AIUTIL (ALTERNATE INDEX CHECK     *
*  UTILITY) TO CHECK FOR, AND SELECTIVELY DELETE AI FILES THAT ARE  *
*  APPENDED TO MAPICS MASTER FILES                                  *
*********************************************************************
// IF DATAF1-?L'229,1'?.SYSCTL          AXZPZ7
//                                      AIUTIL
//                                      SWITCH XXXXXXX0
//                                      LOCAL OFFSET-214,DATA-'?4?'
```

Figure 12-8a

Screen prompt AIDSPFM

```
 00000000            ALTERNATE INDEX FILE CHECK UTILITY                00

 00000000 has the following AI files appended to it.
          If you are going to reorganize this file, delete the associated
          AI file(s) before continuing with the reorganization, or the
          reorganization procedure will crash and burn.

                          A I    F I L E S

                       00000000    00000000
                       00000000    00000000
                       00000000    00000000
                       00000000    00000000
                       00000000    00000000

Do you want to delete these files? (Y=yes, N=no)..... N


 00000000000000000000000000                               CK24 TO CANCEL
```

Figure 12-8b

Screen format member AIDSPFM

```
*        1         2         3         4         5         6         7         8
    SAI01             90 Y                                        Y
    DDATE       8 2 3Y
    DFL0002    34 223Y                                    CALTERNATE INDEX FILE CHX
    DECK UTILITY
    DWSID       2 276Y
    DPARENT     8 5 4Y                  Y
    DFL0005    42 513Y                                    Chas the following AI fiX
    Dles appended to it
    DFL0006    63 613Y                                    CIf you are going to reoX
    Drganize this file, delete the associated
    DFL0007    60 713Y                                    CAI file(s) before contiX
    Dnuing with the reorganization, or the
    DFA0001    45 813Y                                    Creorganization procedurX
    De will crash and burn
    DFL0009    161030Y                         Y          CA I     F I L E S
    DAI1        81228Y
    DAI2        81240Y
    DAI3        81328Y
    DAI4        81340Y
    DAI5        81428Y
    DAI6        81440Y
    DAI7        81528Y
    DAI8        81540Y
    DAI9        81628Y
    DAI10       81640Y
    DFL0025    5319 3Y                                    CDo you want to delete tX
    Dhese files? (Y=yes, N=no)
    DOPTION     11957Y  YA          90                    CN
    DERRMSG    2524 490             9090
    DFA0001    142466Y                                    CCK24 TO CANCEL
```

Canceling MAPICS' AMZ00 Job Automatically

answered by Gary T. Kratzer

Q Because we use MAPICS II, a MRT-NEP security job (i.e., program AMZOO) is always running, making it impossible to run a COMPRESS unattended at night unless someone remembers to cancel the program manually before we leave. How can we cancel this MRT-NEP at a specified time each night so we can run a COMPRESS?

A You can call a MAPICS procedure to cancel the AMZOO security program, letting you successfully run your COMPRESS unattended. Just include the following OCL in your procedure:

```
// LIBRARY NAME-AMALIB
// MEMBER USER1-AMZ09
AMZP01 Z,,,,,,,,,B/N
```

Using Autoresponse When Condensing AMALIB

answered by Mike Patton

Q I have a procedure that performs many system and MAPICS activities in a nightly unattended mode, and I have an autoresponse to handle some of the error messages that may occur. In procedure SS0303 (keysort all index files), I have a statement to condense AMALIB. If this library is in use, the autoresponse answers the message with a 2 option. According to the history report, this procedure is working correctly; however, the job is terminating with this 2 option. Why?

A It is likely that MAPICS is programmed to cancel a procedure if the 2 option is taken in response to the error message you encountered. When the system autoresponds to the error, the return code (obtained via the ?CD? substitution expression in OCL) is set to 3721. This indicates that the controlled cancel option was specified, but not as an autoresponse. To solve your problem, you need to do one of the following: 1) don't condense AMALIB during the keysort procedure, 2) set your autoresponse level to 0 with one of the commands in Figure 12-9a, or 3) disable the automatic response for SYS-2582 by creating response source member ALLOWERR (Figure 12-9b) in AMALIB and then executing the command in Figure 12-9c. The N in column 6 of Figure 12-9b specifies, in the system message member, that no autoresponse is allowed.

Figure 12-9a

Commands to set autoresponse to 0

```
NOHALT 0,JOB        (to disable autoresponse for the job only)
NOHALT 0,SESSION    (to disable autoresponse for the session only)
NOHALT 0,SYSTEM     (to disable autoresponse for the entire system)
```

Figure 12-9b

Response source member ALLOWERR

```
SYS
2582 N
```

Figure 12-9c

Command to disable autoresponse for SYS-2582

```
RESPONSE ALLOWERR,AMALIB
```

Performance

CHAPTER

13

Managing S/36 Performance - Part 1 A Perspective

by Debra Kahn

Code on diskette:
Procedure NEWDISK

Analyze your organization's requirements and system resources using one MIS team's experience in S/36 performance management. The analysis includes methods for determining users' system resource needs and a list of 11 SMF counters for determining system resource use.

How-to advice abounds for S/36 MIS managers interested in improving (or just maintaining) system performance. Most such counsel takes the form of a performance management plan based on a complex system for "trending" or tracking system resource use. Such plans have obvious benefits; for example, they help MIS managers anticipate performance bottlenecks and plan accordingly. However, while most plans are presented with the admonition that they should be adapted to the needs of the individual organization, few plans outline methods for adapting the plan. This article will show you the techniques the MIS team used to tailor the plan to their organization.

In February 1987, the Duke Communications International (DCI) MIS department expanded its 5360 hardware to handle the additional personnel and increased workload precipitated by company expansion. MIS personnel Rebecca Langren and Bob Skowron upgraded the company's Model B to a Model D, enlarged system memory from 1 MB to 3 MB, and increased DASD from 400 MB to 750 MB. This added system spaciousness presented Langren and Skowron with two problems as they tried to keep their system running smoothly: balancing active files, libraries, and folders across three disk drive spindles and adjusting cache sizes so that system memory could be used efficiently.

To solve these problems, Langren and Skowron had to improve their disk and memory management techniques. Monitoring the organization's system resource *requirements* and *use* helped Langren and Skowron decide how best to implement their solutions — and their monitoring techniques can be used in virtually any S/36 shop. Because user needs and expectations provide the context for interpreting system performance data, Langren and Skowron began their analysis by "monitoring" their users.

To gather the user perspective on performance, Langren and Skowron first adopted an active listening policy. Their goals were to keep on top of planned company and departmental expansions that could represent additional system workload, and to get a feel for general satisfaction with system throughput.

As a means to these ends, they devised several formal and informal methods for gathering and documenting company feedback. Informal conversations with users helped MIS gain a general picture of areas of concern; a formal survey and in-house help line helped pinpoint specific concerns and problems and let the users know MIS cared about their opinions. (For guidelines on what kind of information you should gather about your organization, see "How Does Your Organization Define Performance?" on page 348.)

Letting users know MIS cared was important to Langren and Skowron because they wanted to encourage users to become "performance-tuning allies." For example, by educating users about performance-killing practices (e.g., one user using more than one workstation to run disk-intensive queries, and thereby overtaxing the disk), Langren and Skowron could correct them before they became habits. They also hoped to gain broad-based user commitment for their performance-tuning efforts and to become familiar enough to the users to encourage continued user feedback.

Skowron says that such communication efforts in general lessen the likelihood that MIS managers operate in a "system-tuning blind." For example, monitoring user expectations helps MIS personnel avoid an attempt to gain subsecond response time when what users may really want is time to ponder before interacting with the next screen. Skowron warns against wasting MIS resources trying to accomplish something users don't want or need.

Langren and Skowron's informal ways of gathering company performance expectations included attending department and management meetings, walking through departments, and manning an in-house user help line. By regularly attending department and management meetings, Langren and Skowron kept abreast of new projects and growth; they also received first-hand information about how resource-related problems affect productivity. Langren and Skowron walked through each department to observe how users work with the system and to give users face-to-face access to MIS for discussing system resource problems and concerns. The in-house help line achieves the goal of easy access to MIS more formally. Because the help line lets users communicate concerns quickly and easily, MIS can track day-to-day problems so they don't fall through the cracks.

Langren and Skowron's most formal method of gauging user concerns, a semiannual MIS survey, had a twofold purpose: to obtain specific feedback on performance areas that informal discussions had indicated were concerns, and to provide a basis for deciding how to expand or change certain user support programs.

Langren and Skowron hoped to use the survey to pinpoint potential bottlenecks, to see how their management of the expanded system resource affected end users, and to determine which direction future resource management should take. Langren and Skowron also wanted to collect input on other MIS-related activities (e.g., system education, problem resolution, and project planning) to gain a measurement of how well MIS was functioning as a department within the company. Finally, they hoped to present the survey results to top management as formal documentation of system resource needs.

Figure 13-1
MIS survey

```
To:    All Staff
From:  Management of Information Services (Bob, Deb & Rebecca)
Re:    MIS Survey
Date:  December 10, 1987


    Please assist us in improving our service to you by responding to
the following survey questions.  We appreciate your thoughts and
comments on the following MIS areas in our company.

    Please rate the following on a scale of 1 - 4, EX or NA:

       1  = Unsatisfactory
       2  = Average
       3  = Good
       4  = Excellent
       EX = Didn't know feature existed
       NA = Do not use feature

______________________________________________________________________
______________________________________________________________________

Please Rate the Availability of S/36 Resource:

___    Access to the S/36 for interactive work

___    The S/36's interactive response time

___    Access to the computer room to pick up your printed reports

       Please evaluate the amount of time you wait for your printed
       reports to come out on the following printers:

___    P1 - Fujitsu band printer

___    P2 - GBT/GE dot matrix

___    P3 - IBM letter quality

___    P6 - Editorial Virtual at Trish Frease's Desk

___    P7 - Editorial Virtual at Jeanne Tatum's Desk
```

Although the multipage MIS survey investigated many areas, Langren and Skowron dedicated the first page to questions about system throughput, the basis on which users judge system performance. The questions on the first page of the survey (Figure 13-1) targeted three areas that potentially could affect users' perceptions of system throughput: system access, system output access, and system output speed.

The first question on the survey focused on the system access area. For example, concerns about wait time for CRT access were expressed by personnel in the marketing department (these personnel did not each have an individual CRT). The second question gauged satisfaction with interactive throughput speed; it reflected directly on how well Langren and Skowron had tuned the system. The third question uncovered concerns about computer-room access and assured Langren and Skowron that location and availability (the system printers were located in the computer room, which was locked except during business hours) weren't clouding users' perceptions of how well the system actually delivered throughput. Finally, the questions about printer wait time helped Langren gauge satisfaction with current job queueing and print spooling methods and pinpoint where output might be resource-bound.

To guarantee that user responses to such questions would be well thought out, Langren visited each department to explain the purposes of the survey. She reinforced MIS concern for department problems and assured users that MIS would use the results of the survey to improve each department's work situation. Langren also found that, as a side benefit, introducing the survey to each department provided an informal forum for airing complaints.

DCI's survey results showed that most users were satisfied with throughput and that problems were limited to specific departments. As predicted, in response to the survey's first question, the marketing department rated system access "unsatisfactory," but dissatisfaction seemed limited to that department. The circulation department's dissatisfaction with response times was also predictable because circulation does a lot of interactive work, some of which involves examining customer records interactively while a customer is on the phone. Finally, the survey showed that although access to the computer room was satisfactory, output bottlenecks existed at the company's letter-quality printer and the editorial department's two virtual printers.

Initially, Langren and Skowron examined whether they could solve the problems users expressed in the survey by changing their resource management (although they realized immediately that solving the output problem might involve purchasing additional printers). To evaluate their present resource management, they began analyzing system use by tracking their System Measurement Facility (SMF) reports. Most IBM experts agree that regularly running SMF and "trending" or tracking the results are necessary components of good performance management.

These IBM experts suggest that S/36 managers track 22 SMF counters: those that monitor the S/36's workhorses (the main storage processor — MSP — and control storage processor — CSP), the system's slowpoke storage facility (the disk), and the system scratchpad (memory). (For more information about these S/36 components, see "Counting on Good (S/36) Architecture" on page 349.) Langren and Skowron accepted this theoretical base; but because they had limited time to monitor the counters and had sufficient knowledge of S/36 architecture, they decided to track a practical set of SMF counters that provided significant system performance information and could be interpreted quickly and easily.

Langren and Skowron chose 11 SMF counters that trace the performance impact of the processors, disk, and memory: MSP Utilization, CSP Utilization, Task Work Area (TWA) Extents, Disk Seeks Greater Than 1/3, Disk Utilization, User Area Disk Activity, Storage Releases L3 and L4, Cache Size, Cache Page Size, Cache Utilization, and Cache Hits and Misses. Langren also developed an automated SMF procedure that simplified monitoring these counters. Each counter provides significant performance information in a key area.

MSP Utilization and CSP Utilization

The MSP and CSP summary counters reveal the utilization of these processors (by reporting the percent of time that the processors are not idle) and tell you whether the workloads are balanced between the two processors. Balanced percentages for MSP and CSP use mean that processor loads are near optimum utilization. High levels of activity in either processor may adversely affect response time.

TWA Extents

The TWA is a system work space on disk, which is the slowest of the S/36 system resources. When a user program is initiated, the CSP assigns a space in the TWA to hold a program when it is paged out of main storage. If the TWA size is insufficient, it remains extended until the next IPL.

TWA extents usually are performance killers because they are not adjacent to the originally configured TWA — they may be on the other side of the disk. Such placement means that performance is slowed by the sluggish, mechanically dependent disk as it moves from TWA to TWA extent, seeking a program that has been paged to disk.

To control TWA extents, Langren and Skowron try to IPL as soon as possible after an extent. They also increased the size of the TWA to accommodate the company's increased use of IBM office products (i.e., DisplayWrite/36 and Query/36), which often cause TWA extents.

Disk Seeks Greater Than 1/3

The value "Disk Seeks GT 1/3" tells you during what percentage of the day's disk accesses the disk arm traversed more than 1/3 of the tracks. These kinds of disk accesses can be performance killers because of the time involved in positioning the disk arm (positioning the disk arm consumes 75 percent of disk access time). The farther the arm must move, the slower the access time and the greater the cost to system performance. Thus, this counter indicates how effectively files, libraries, and folders are placed on the disk spindles.

Disk Utilization

The Disk Utilization snapshot value reveals how long during the SMF interval each disk was busy. Langren and Skowron use this counter as a measurement of how well they have balanced workloads across their disk spindles. A well-balanced workload should reflect nearly equal utilization percentages across the system spindles.

User Area Disk Activity (UADA)

The UADA counter (new with Release 5.1 of the SSP) is a summary value that reflects virtual paging activity to and from disk and reflects translated transfer (transient) loads (i.e., when SSP programs are invoked). The UADA represents the shuffling of system and user programs and block data between main memory and disk when main storage is overcommitted. Most experts agree that the UADA counter is the best indicator of how efficiently main storage is being used. Langren and Skowron watch this counter closely because when it gets too high the system spends more time paging things in and out than working.

Storage Releases L3 and L4

Storage releases L3 and L4 are the best indicators of whether you are overtaxing main storage. Storage release levels (L1 - L4) indicate when one program has preempted another in memory. L1 and L2 storage releases, which indicate that the preemptor has a higher priority than the preempted, should not cause concern — but L3 and L4 storage releases should.

An L3 storage release indicates that the priority of the program paged into main storage was only slightly greater or equal to the program that was paged out; an L4 storage release indicates that the program paged out had a higher priority than the program paged in. The L3 and L4 storage releases occur when main storage is so overloaded that the system is forced to preempt important programs just to make sure lower priority programs make some progress toward completion.

Cache Size and Cache Page Size

Cache Size is a measurement of the slice of memory you set aside for buffering data through S/36 Cache. Its value is greater than zero only when you have turned on S/36 Cache, and then it must be at least 64 K (maximum value is no more than the size of the user area in main storage). Cache Page Size, another value you set when you engage S/36 Cache, is the smallest unit of data that S/36 Cache can bring into memory. Each cache page must be at least 1 K but not more than 16 K, and the ratio between cache page size and cache size must result in at least 32 pages of cache.

These values affect performance when contention for memory space is great. If too much memory is allotted to caching, the virtual page rate (the UADA) may increase enough to degrade system performance because user programs are contending for a smaller user area in memory. If too little memory is allotted to caching (or cache pages are too small), then system performance may not gain the full benefit of cache's performance-enhancement capability because the data moved into memory is insufficient to minimize disk accesses.

Cache Utilization and Cache Hits and Misses

Cache utilization is the percentage of cache reads that were found in memory (i.e., data for which the system did not have to access the disk). It is calculated using the Cache Hits and Misses ratio. A cache hit occurs when the system finds a needed record in the cache. A cache miss occurs when the system does not find the record in the cache and must read the disk instead. Langren and Skowron monitor the cache utilization counter to measure cache's positive effects on system performance — using it as a yardstick to measure whether their cache sizes were set to gain the most performance benefit from the cache facility.

Automated SMF Procedure

Because SMF should be run on a regular basis before the counters are used to make performance management decisions, Langren created a procedure that runs SMF automatically. The procedure produces daily summary and detail reports (see Figure 13-2) and tailors the SMF snapshot-taking process to her needs.

Figure 13-2
NEWDISK procedure

```
// TAG AGAIN
SMFSTART 500,200,,N,SMF.LOG,,,,,Y
// WAIT INTERVAL-001500
SMFSTOP
SMFPRINT ALL,Y,P5,SMF.LOG              SEND TO P5
// IF ?TIME?>171500 DELETE SMF.LOG,F1
// IF ?TIME?>171500 CANCEL             OFF AT 5:15PM
// GOTO AGAIN
// RETURN
```

Procedure NEWDISK, designed at a time when Langren was tracking the need for additional DASD, uses the three SMF commands SMFSTART, SMFSTOP, and SMFPRINT to accumulate SMF information at 15-minute intervals during the day, print both summary and detail reports (indicated by the ALL on the SMFPRINT command) at 5:15 p.m., and then begin again. (For more information about these SMF commands, see Chapter 2 of the *SMF Guide*, SC21-9025.) The procedure allowed her to accumulate information about how the system was being used throughout the workday (8 a.m. to 5 p.m.), as well as follow her system's nighttime workload.

Langren's plan proved to provide adequate measurements — despite some experts' recommendations for shorter intervals — because the system was in constant use during the nine-hour workday (during which there were no real peaks or lulls in interactive use) and because some batch work was done at night (which allowed her to compare daytime batch processing data to nighttime processing data).

Using the Information

From the perspective she gained by regularly monitoring the 11 SMF counters, Langren began to address her organization's expressed concerns. She could address the circulation department's concerns about response times by continuing to improve disk and memory maintenance. By monitoring some secondary SMF counters, she realized that the marketing department's problem with CRT access could be managed with job scheduling techniques and improved communication. The printer output problems could be similarly alleviated, but Langren and Skowron hope to purchase a second letter-quality printer eventually, so individual correspondence can be separated from batch-generated form letters.

With the perspectives provided by ongoing organizational and system monitoring efforts, an MIS manager can make knowledgeable decisions about system performance management. Although IBM experts have provided threshold values for S/36 resource use (see "S/36 SMF Threshold Values: A Thumbnail Guide" on page 351), the information must be tempered with knowledge of current system performance and expected system performance to properly apply performance-tuning techniques and make decisions about hardware upgrades.

Sometimes the best performance technique is to ignore a threshold value and leave a system alone if it is performing to user satisfaction. If you anticipate an increase in system workload, begin planning for additional resource capacity. In DCI's case, Langren and Skowron needed an ongoing maintenance plan to keep carefully placed files and folders together and balanced across three disk spindles, and they needed some type of decision mechanism for maintaining efficient cache sizes.

In parts 2 and 3 of "Managing S/36 Performance," you will see how Langren and Skowron used a file placement and disk balancing program to yield acceptable disk utilization levels and keep disk seeks greater than 1/3 to a minimum and, in addition, managed file placement so that space for the growing files, libraries, and folders related to new projects could be anticipated. You will see how building a decision matrix for cache sizes helped them adjust memory to handle current workload efficiently and provide a path for adjustment to additional workload.

Although Langren and Skowron's specific placement of files on their spindles and their chosen values for cache sizes may not be universally applicable, the methods by which they approached these decisions, described in the next two parts in this series, can be applied in any S/36 shop.

How Does Your Organization Define Performance?

Bob Skowron of the DCI MIS team suggests tailoring performance management to an individual company by first establishing a clear picture of how your company defines system performance. Although a complete definition of performance should include an understanding of the internal workings of the system, MIS managers can begin understanding how their organizations define performance by answering three questions about their system's external environment.

What is the nature of the organization?

In its simplest form, answering this question reveals whether the expected system workload is interactive, batch, or mixed. Workload in a mixed environment, such as exists at DCI, presents a different performance management challenge (e.g., careful job scheduling techniques for batch work) than for exclusively interactive or batch environments. In addition, answering this question can reveal organizational characteristics that help direct performance-tuning activities: is the business customer service oriented or analytical, is it fast paced or slow paced, does it have a simple or complicated personnel structure?

How do users perceive your system's public behavior?

In addition to surviving within a certain organizational structure, your system also has to please its public. In a mixed environment, there are probably as many definitions for system performance as there are users. For example, a customer service representative may define system performance as how fast the screen comes back after entering information into a customer record (interactive response time), and the accounting department director may define performance as how long a general ledger takes to post its accounts (batch processing time). Both definitions of system performance are valid and should be accounted for in an overall performance management plan.

What are the day-to-day throughput expectations of the organization?

Every true-hearted MIS manager knows that what users want and what is practical are often two different things. A customer service representative may tell you he wants subsecond response time when, in fact, he could easily tolerate a two- or three-second response time without adverse affects on service calls. However, if the sales department has a data-entry process that closes down at 5 p.m., and your system must summarize that process for a communications transmission scheduled for 7 p.m., that batch program *has* to finish in two hours; there is no compromise. So, the final piece to the puzzle of defining system performance is an examination of the level of activity necessary to keep daily, weekly, monthly, and yearly business transactions running smoothly.

Counting on Good (S/36) Architecture

by Mel Beckman

To understand better how a selection of 11 S/36 SMF (System Measurement Facility) counters can describe system performance, MIS personnel may want to refresh their knowledge of S/36 architecture. The following discussion reviews some important aspects of S/36 architecture.

Generally, S/36 architecture, which evolved from IBM's single-processor System/3, supports two dissimilar main processors, several subordinate processors, a disk storage system, and main storage, which itself is divided in function. Understanding the relative speeds and functions of each of these components prepares you to interpret information you eventually gather about resource usage.

To begin, the "thinker" in S/36 architecture, although a slow one, is the Main Storage Processor (MSP). The MSP runs SSP programs and user applications by using a S/3-based commercial (memory-to-memory) instruction set. S/36 main storage has a 23-bit address, which means it has an 8 MB potential for work capacity (currently, the largest 5360 — the Model D — has a 7 MB main storage capacity). But the MSP has no control over which programs are being run in main storage; it performs only computations — at a relatively slow 0.6 million instructions per second (MIPS), or 1.6 microseconds per instruction.

In comparison, the Control Storage Processor (CSP), the "go-fer" in S/36 architecture, works twice as fast as the MSP; it executes 1.2 MIPS, or 800 nanoseconds per instruction. The faster speed is a result of its more suitable register-to-register minicomputer instruction set. The S/36 CSP interfaces with peripheral devices, manages main storage contents (the virtual storage mechanism), and controls the execution of the MSP. The CSP controls all input and output (I/O) and tries to keep the MSP operating at maximum efficiency. The CSP has a 16-bit address and can contain either 64 K or 128 K of control storage (its own memory), depending on whether it also must function as an inboard Workstation Controller (WSC).

In addition to the MSP and CSP, a S/36 also can contain a third kind of processor, the CSP/I, which has the same instruction set and organization as the CSP, but acts as a dedicated controller for certain I/O operations. The WSC, the Eight-Line Communications Adapter (ELCA), and the Data Storage Processor (DSC) contain one CSP/I each. The data path used by the MSP, CSP, other processors, and peripheral devices is known as the I/O Channel and is controlled by the CSP. The I/O Channel is an intelligent device that provides the I/O transfers (10 MB in two directions) between the disk and main storage, as well as a high-speed path between the CSP and MSP.

The complete multiprocessor structure of the S/36 can move data faster than conventional single-processor CPUs. Remember that a single-processor architecture (e.g., the IBM S/3) has only one intelligent device to handle all types of processing: user programs, I/O, scheduling, and memory management. The S/36, by contrast, splits these functions among different intelligent devices so that many of these tasks can be handled simultaneously. But as fast as this processor structure is, the S/36 still is dependent on a slow

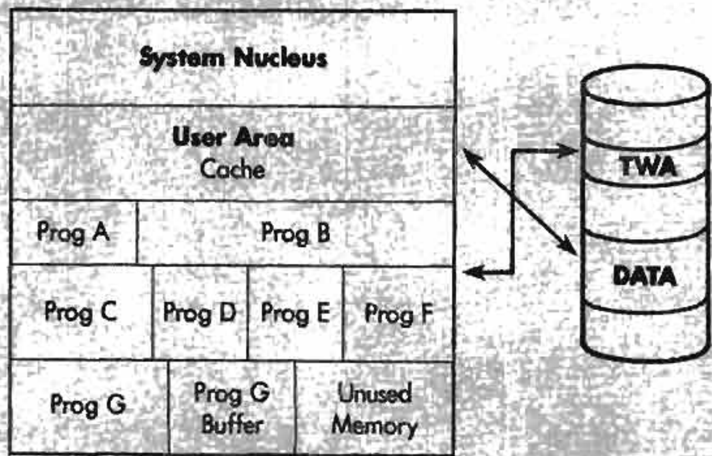
Continued

data storage and retrieval mechanism, the disk spindle, with its ever-worrisome mechanical parts. Average disk access time (the time it takes the mechanical parts to find and retrieve data) is 35 milliseconds, which makes it 21,600 times slower than the MSP.

The S/36 has a 24-bit address for disk storage, which means it has the potential of addressing *4,096 MB of disk or 20 200 MB disk spindles*. Of course, that much S/36 disk storage is not yet possible and managing even one spindle is a challenge.

The speed and capacity of the processors and disk represent only one side of the performance story of the S/36; the other side is memory capacity. S/36 main storage comprises primarily two parts, the system nucleus and the user area (Figure 13-3). The system nucleus contains work space for the SSP and other IBM-supplied system operation software. The user area contains space for user programs, S/36 cache, and user program buffers. It is important to remember that all these areas compete for a portion of the same resource.

Figure 13-3
S/36 main storage



Data that has been "blocked" for a particular program (e.g., Program G in Figure 13-3) is copied from disk into a portion of the user area that will be used exclusively by that program. In contrast, data that has been "cached," although it also is copied from disk into the user area, can be accessed by any active user program. Thus, the relative merit of using S/36 cache when you have many user programs accessing the same file is obvious. (IBM's June 1986 announcement of the 5360 Model D included an expansion of main storage capacity from 2 MB, then available on the Model C, to 7 MB. Concurrently announced memory expansion cards in 1 MB or 2 MB capacities provided Model C owners an upward growth path, but also precipitated some confusion regarding how best to take advantage of the additional space with S/36 cache.)

Continued

The S/36 memory management scheme tries to keep main storage as full as possible at all times. Space that isn't taken up by user programs is used to hold frequently used SSP programs — thus the S/36 is always doing some virtual paging (UADA). Blocked data is kept in "virtual" storage and thus becomes a candidate for paging whenever user area storage is needed for another task. However, cache pages are handled separately because whenever records in cache are updated, they are immediately written to disk. When cache pages are needed for more important records, their current contents are simply discarded. User programs are paged out to the TWA on disk when user area storage is needed by another task. When a program has been paged out to disk, its execution is slowed because it must first be paged back into memory before resuming execution.

S/36 SMF Threshold Values: A Thumbnail Guide

In past articles, IBM performance experts Ken Willkomm, Jeff Dixon, and Ray McRoberts have recommended threshold values for judging system performance based on system measurement facility (SMF) data. This summary of their recommendations for the 11 SMF counters chosen by DCI MIS personnel Rebecca Langren and Bob Skowron will help you interpret SMF data correctly as you plan your S/36 performance tuning; use it as a guide to possible performance solutions based on SMF data. You, of course, must take into account the complex factors that make up your data processing environment before you consider implementing the performance-tuning suggestions this summary offers. Even when SMF values indicate possible performance hazards, you will need to consider how frequently those values occur and whether the potential problems they indicate affect system performance.

MSP and CSP Utilization Percentages

Main Storage Processor (MSP) or Control Storage Processor (CSP) utilization over 60 percent, when coupled with slow interactive response time, may indicate that your system needs performance tuning. When MSP utilization is high in a sluggish system, you probably should reschedule MSP-intensive activities (e.g., sorts, compiles, multiple batch jobs). (To determine which tasks are MSP-intensive, examine your SMF snapshot report for tasks that show high percentages for I/O and System Event Counters.) When CSP utilization is high in a sluggish system, you should reschedule CSP-intensive programs, such as BASIC, BRADS, or FORTRAN, and examine your CSP's workload to determine whether it needs a Data Storage Controller, Eight-Line Communications Adapter, or additional workstation controller. If your "trending" mechanism indicates that processor use will continue to increase beyond 60 percent, you should consider a processor upgrade (e.g., to the Model D processor).

Continued

Task Work Area Extents

You should consider increasing the size of the Task Work Area (TWA) when Task Work Area Extents show up on SMF summary reports. It is not unusual to increase the TWA to several times its previous value when IBM office products are installed (for example, DCI MIS personnel increased their system's TWA from 1,200 blocks to 3,000 blocks because of the company's intensive DisplayWrite and Query use). The TWA size can be changed by using CNFIGSSP and then performing an IPL. (See *Changing Your System Configuration*, (SC21-9052).)

Disk Seeks Greater Than 1/3

Because disk accesses that traverse more than 1/3 of the tracks can quickly degrade system performance, this SMF summary value should be kept as close to zero as possible. To accomplish this objective, group together a disk's most frequently used files, folders, and libraries to reduce the distance the mechanical arm must travel during a disk seek. There are several prescriptions for placing objects on a disk, all of which recommend incorporating free space on the disk to anticipate file and folder extents. Reducing disk seek distances can increase the number of disk operations performed during a given time and thus prevent the disk accesses from becoming a performance bottleneck.

Disk Utilization

For best overall performance, the SMF summary disk-use values for all spindles should be balanced within 10 to 15 percent of each other. The lowest utilization percentage should occur on disk A1, which contains the system software and work areas. A relatively high utilization percentage for one disk indicates unbalanced disk loads or poor file placement. To reduce disk utilization percentages in general, consider one of the previously mentioned file placement methods, as well as engaging cache or altering its parameters, examining (for reduction or elimination) alternate file indexes and blocking, and rescheduling batch jobs.

User Area Disk Activity (UADA)

An average UADA of 200 pages per minute or less is nothing to be alarmed about — if response times are respectable. Sluggish response times may indicate a high virtual paging rate, which can be lowered by reducing cache size and rescheduling some batch jobs. If your average UADA is between 200 and 400 pages per minute and response times are sluggish, your system's main storage may have insufficient space for all its tasks. Reducing cache size and rescheduling some jobs may help improve response time, but you probably need to begin capacity planning. When the UADA goes above 400 pages per minute, increasing main storage helps reduce response times and increase throughput.

Continued

Storage Releases L3 and L4

In addition to the UADA level, storage releases L3 and L4 may indicate insufficient memory capacity. If your total number of L4 releases is below 20, you should apply some of the previously suggested performance-tuning methods for improving memory use. If your L4 releases are above 20 and your UADA activity exceeds 400 pages per minute, you may need additional resource.

Cache Size and Cache Page Size

Because cache pages are shared among users, the system uses memory more efficiently. This efficiency should be reflected in lower UADA rates when cache size and cache page sizes are set properly. The cache size and cache page size parameters are set with the CACHE procedure. The minimum value for cache size is 64 K, and the default value is one-fourth the size of main storage. The accepted values for cache page sizes are 1 K, 2 K, 4 K, 8 K, and 16 K; the default is 2 K. You must specify a cache page size value that allows at least 32 pages to be created in cache.

Cache Utilization and Cache Hits and Misses

An average cache use below 40 percent could mean that cache size is overallocated or is not needed at all (adjusting cache and cache page sizes and continuing to monitor cache utilization will help you decide which is true). Most experts say that average cache use should be at least 65 to 75 percent before system performance is benefiting from the facility. In addition, your total cache hit-to-miss ratio probably is at least 2 to 1 before you can see any significant improvement in response time. If your SMF summary report reflects a hit-to-miss ratio of less than 2 to 1, you should adjust the relationship between cache size and cache page size to better accommodate your job mix.

Managing S/36 Performance - Part 2
A Streamlined Approach to S/36 Disk Management

by Debra Kahn

Code on diskette:

Procedures VDSKTOA3, FLDCMPP, STMBP01, STMBP02
RPG program FLDCMP

Learn how to organize and maintain your disk space.

In general, Bob Skowron and Rebecca Langren's S/36 performance management goals for Duke Communication International (DCI) were the same as those of all S/36 MIS managers: to manage disk and memory resources effectively so the system performs satisfactorily and the MIS team can plan for and accommodate increased use. But as Langren and Skowron attempted to implement IBM's recommended methods of disk management, they began to encounter problems. This article describes how Langren and Skowron handled these problems by improving recommended disk management practices for object placement, free space location, and disk space balancing. Their solutions can be adapted for use in any S/36 shop.

Langren and Skowron's initial disk-tuning activities centered on organizing files, folders, and libraries on individual disk spindles for better performance. As the previous article explained, the organization of objects on a spindle plays a crucial role in system performance because it affects the amount of time the mechanically dependent disk arm takes to locate different objects. If objects on a disk spindle have been well placed, the disk arm will not have to travel far between requested objects, and disk accesses will not slow system performance significantly. If objects have not been well placed on the spindle, however, the disk arm must traverse a greater distance between the objects it is seeking. When many such accesses are made in a small amount of time, the relatively slow nature of this type of disk access affects performance.

This performance-degrading disk behavior is measured by the System Measurement Facility (SMF) counter Disk Seeks Greater Than 1/3. The counter indicates the number of disk accesses during which the disk arm had to traverse more than one-third of the disk tracks. Thus, one objective of careful disk space organization is to reduce or eliminate the disk seeks greater than one-third during the system's work day.

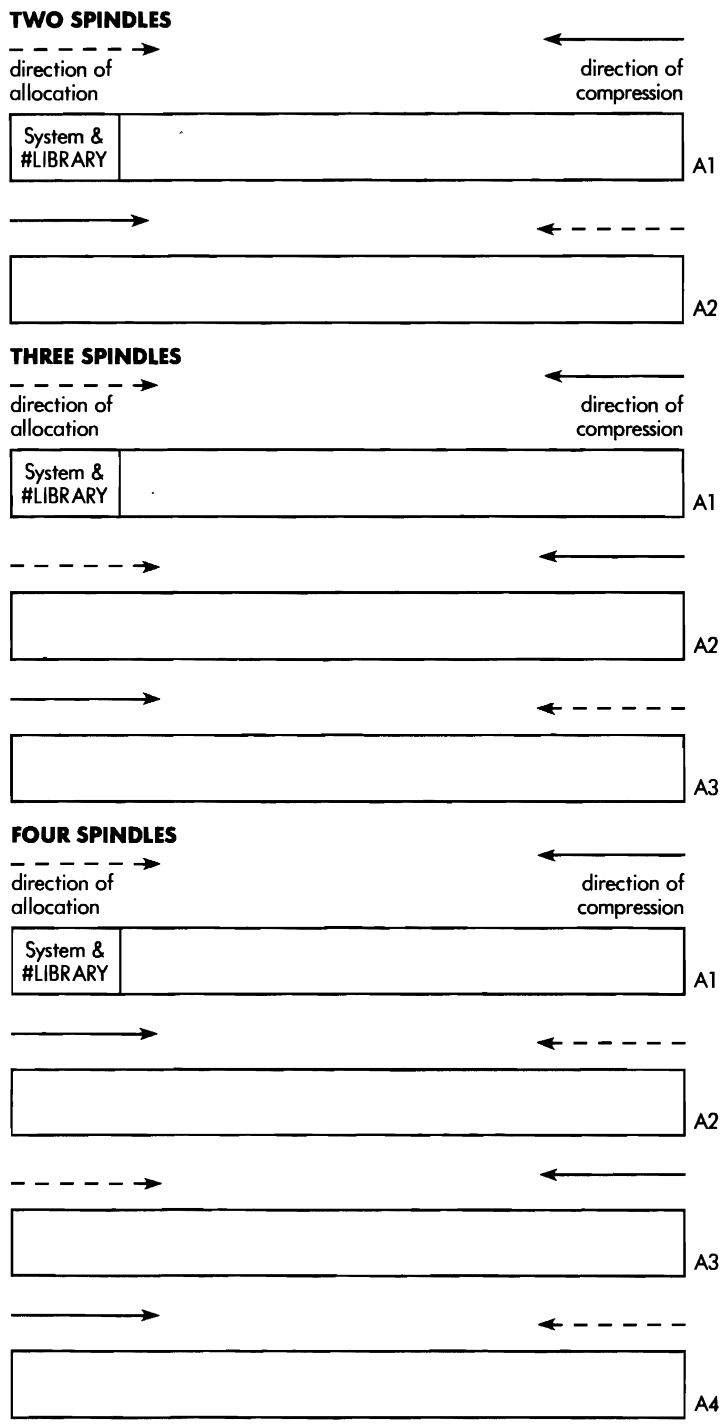
Langren and Skowron encountered two problems as they initiated IBM's recommended methods of disk space organization on their individual spindles. Langren and Skowron's first problem was inherent in their addition of a third disk spindle: the direction of disk compression on the second spindle in a three-spindle configuration is opposite from the direction of disk compression on the second spindle in a two-spindle configuration. They questioned whether they should follow the manual's recommended change in disk organization for the second spindle. Second, Langren and Skowron's original experience with recommended disk space organization methods, as practiced on their two-spindle system, had shown that the recommended location of work space (free space) on disk spindles accomplished by simple disk compression methods tends to allow increases in disk seeks greater than one-third instead of controlling them. The two problems actually were interconnected; to understand them better, let's examine the recommended steps for object placement.

IBM's Object Placement Scheme

Both Chapter 4 of the *S/36 Concepts and Programmer's Guide* (SC21-7903) and IBMer Ken Willkomm present guidelines for the logical placement of data on a disk spindle based on its direction of compression and the resulting location of free space. The direction of compression for a particular spindle is determined by its position in a one-, two-, three-, or four-spindle system configuration (Figure 13-4). The direction of compression is opposite from the direction that the disk arm moves as it allocates space for a new file.

Figure 13-4

Directions of compression and allocation for S/36 spindle configurations

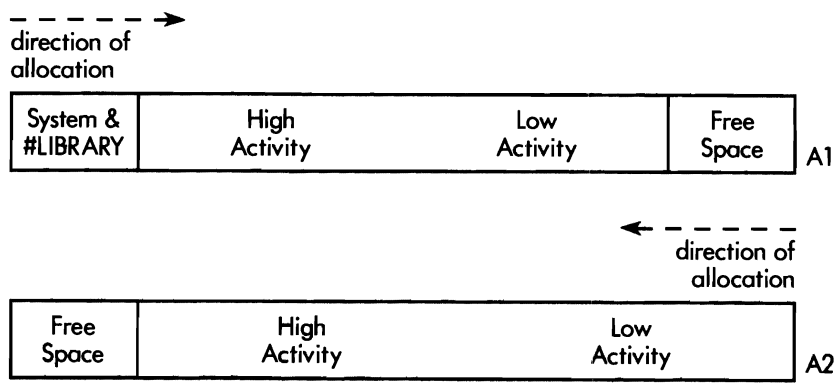


When the system is told to allocate an object on a particular spindle, it searches for the first available, adequate space. As objects are created, deleted, reorganized, and extended, disk space on a spindle may become fragmented, leaving little contiguous free space for new objects. To help you free up space on a disk spindle, IBM has supplied the S/36 COMPRESS command. With this command, you can collect contiguous free space on a targeted disk and even initiate a process to organize objects on that disk.

To aid in the process, the command's optional second parameter lets you specify the desired location for the area of contiguous free space (i.e., the highest block numbers or the lowest block numbers) on the spindle. Thus, the COMPRESS A1,FREELOW command collects the available free space at the lowest block numbers (the end at which the disk arm begins movement during file allocation) of spindle A1. If the second parameter is not used with the COMPRESS command, the system will collect the free space at the "high" end of spindle A1 and at the "low" end of each of the subsequent disks. (For more information about the COMPRESS command, see Chapter 4 of the *S/36 System Reference manual* (SC21-9020).)

The free space location parameter of the COMPRESS command figures strategically in IBM's recommended procedures for arranging objects on a disk. In fact, the recommended procedure begins with a FREELOW compress of spindle A1. This step frees space next to the system files and #LIBRARY (always located on the low end of spindle A1). IBM then recommends that you move files, folders, and libraries into this free space by using the appropriate commands; the most frequently used files should be placed next to #LIBRARY. A second compress, this one with FREEHIGH specified, draws the files, folders, and libraries together and frees space at the high end of the disk. By applying these steps to A1, you move the most frequently used user files, folders, and libraries next to the frequently used system files and #LIBRARY; thus, the disk arm will not have to move far between the system objects and high-activity user objects, and disk seeks greater than one-third should be reduced. In addition, spindle A1 will contain contiguous free space where new objects, as well as file and folder extents, may be placed.

Figure 13-5 *IBM's recommended disk space organization*



The recommended procedure for freeing space and organizing objects on subsequent spindles is simpler because those spindles do not contain system objects. IBM first recommends that the COMPRESS command be used without the free space location parameter, thus creating free space at the low end of the disk. Then, files, folders, and libraries should be moved into the free space before executing another COMPRESS. These operations result in the most frequently used objects on all spindles — except A1 — always being located next to the available free space (see Figure 13-5 for a two-spindle system configuration). The benefits to this placement plan are simple execution and the allocation of new object, file, and folder extents (which often become most frequently used objects themselves) next to existing high-activity areas on spindles A2, A3, and A4. Thus, the placement reduces the distance the disk arm must travel from the most frequently used files to newly allocated files or extents.

The Drawbacks to IBM's Object Placement Scheme

Although IBM's recommended procedure considers and uses the direction of compression on a spindle, it neither considers how the resulting free space may be used nor considers the direction of arm movement across the spindle during object allocation. As a result of these oversights, the disk spindles can untune themselves quickly as extents and new objects are introduced, leading to performance degradation.

To better understand the impact of these problems on system performance, let's examine a hypothetical situation involving file extents, new file and folder allocations, and file reorganizations on a recently compressed two-spindle system. When allocating a file, the system begins searching from the end of the preferred drive in the opposite direction of compression. If it finds space that can hold the file being allocated, it puts the file there. If the system does not find space on the preferred drive, it searches the other drives for space to allocate the file (or the remainder of the file).

To begin our hypothetical situation, suppose three of the high-activity files on spindle A1 need extents, as is often the case. Because the disk has just been compressed, the system can find adequate, available space for the extents only in spindle A1's free space. At this point, a possible performance problem is already apparent: a disk access of the extended files most likely will have to span more than one-third of the spindle tracks because the free space on spindle A1 is located on the opposite side of the disk from where the original high-activity files are placed.

Next, a user creates a new DisplayWrite/36 (DW/36) folder on spindle A1. Under our hypothetical circumstances, the system's tendency to place the folder in the A1 free space can result in a disk seek greater than one-third during subsequent accesses to the folder. That is, the distance between the new DW/36 folder and a previously accessed DW/36 folder or DW/36 system file usually is greater than one-third the distance of the tracks.

Finally, suppose a new, low-activity file has been allocated to spindle A1. Again, the file is placed in the free space on A1. But as a result of the disk arm's direction of movement during file allocation and the location of the free space, the first available, adequate space for the file likely is after (i.e., to the right of) the high-activity file extents and the new DW/36 folder.

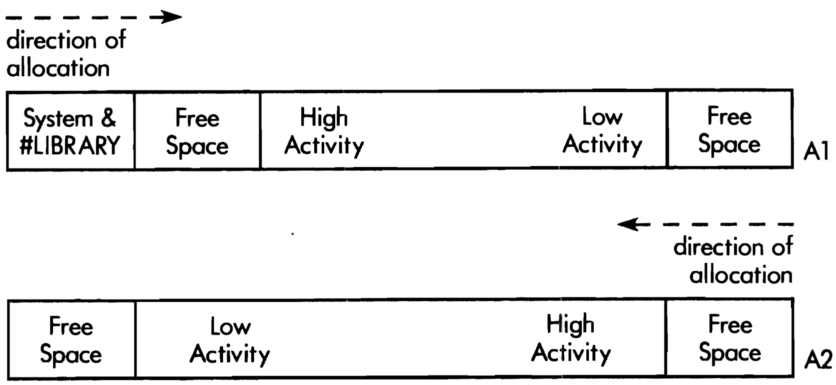
Our hypothetical file allocation and earlier operations would result in the following jumbled organization for spindle A1 (from low to high end): system files and library, high-activity files and folders, low-activity files and folders, three high-activity file extents, one new DW/36 folder, and one low-activity file. If we add to our hypothetical sequence a couple of reorganizations of high-activity files on spindle A1, these files' resulting shift to the free space intensifies the movement of our spindle's organization toward randomness.

A similar jumble eventually would result on spindle A2 as low-activity files are extended and new, high-activity files are added. As a result, not only would the recommended disk organization of our two-spindle system quickly be thwarted, but disk seeks greater than one-third would gradually increase as the disk becomes disorganized. Only a disk compress and a time-consuming manual reorganization of objects on the spindles would restore order to the disks and good performance to the system. Any good disk management plan recommends frequent disk compresses, but following IBM's recommended plan for disk space organization requires unnecessary diligence and labor. IBM's plan also requires a dedicated system because objects cannot be moved when they are in use.

A Better Compress

A contributing factor to the inadequacy of IBM's recommended disk organization plan is the COMPRESS command's free space location parameter, which allows for only one area of free space on a disk spindle. The IBM plan tries to overcome this limitation by recommending that high-activity files be placed next to this contiguous free space, but as you saw in

Figure 13-6
New disk space organization



our hypothectical case, such placement eventually can create disorder. To solve this problem, Skowron reasoned that S/36 disk spindles could be organized in a way similar to RAM (Random Access Memory) on a PS/2: by inserting free space between objects.

Skowron's plan (shown in Figure 13-6 for a two-spindle system) places free space at both ends of each disk spindle and reverses the recommended organization of objects on all spindles except A1. The plan not only allows more reasonably located object extents, but encourages the disk to self-tune during object allocation. These benefits accrue from consideration of one, the direction of disk arm movement during object extents and allocations, and two, the likely uses of free space on a spindle. Let's examine how Skowron's plan addresses these considerations by applying our hypothetical situation to a two-spindle system organized according to Skowron's plan.

Skowron's plan first places one area of free space between the system files and the high-activity user objects on spindle A1. Thus, when our hypothetical high-activity files extend, the extents likely will be placed in this free space because it is the first adequate, available space the disk arm will encounter as it searches for space for the extent. Because the distance between the high-activity files and their extents is minimal, this placement is more desirable than that provided by IBM's recommended disk space organization. Subsequent disk accesses of these files should not involve disk seeks greater than one-third of the disk tracks.

When our hypothetical user creates his or her new DW/36 folder, it likely will be placed in the first free space also. Again, this placement is more desirable than that provided by the IBM plan because the distance between the DW/36 system files, the new DW/36 folder, and the other DW/36 folders is minimal. Subsequent use of the new DW/36 folder also should not result in disk seeks greater than one-third. Thus, the creation of free space at the low end of spindle A1 greatly enhances system performance by reducing disk arm movement (and thereby reducing access time) during certain types of disk accesses. As an added performance benefit, Skowron's plan for free space at the low end of spindle A1 places spool file extents next to the system files for easier access.

Skowron's plan also creates free space at the high end of spindle A1. This second free space works in tandem with the first to encourage the disk to tune itself. Self-tuning occurs because the second free space is used for new objects and extents only when the first free space has been used up. Because frequently used objects usually are extended or allocated first after disk compression, they stay in the high activity area on the spindle; similarly, the least-used object extents usually will go to the second free space. In addition, natural attrition of objects works with the disk organization and disk arm movement during allocation to keep the least-used objects at the high end of the disk. Holes created by deleting objects usually are filled with objects of similar importance during allocation or are compressed out; thus, least-used objects usually drift toward the high end of spindle A1 as a result of normal disk maintenance.

Finally, Skowron's plan organizes subsequent spindles similarly to A1's organization: free space and high-activity files are located at the end of the disk where disk arm movement begins during file allocation. This organization is the reverse of IBM's recommended organization for these spindles. In addition to having the same advantage as IBM's organization — keeping high-activity file extents near the original files — this new organization for spindles A2, A3, and A4 has the same self-tuning advantage offered by the new organization of spindle A1. The spindle organization works with the system's allocation strategy to encourage self-tuning on the spindle if you follow simple maintenance practices.

The maintenance tool that helps make Skowron's disk space organization plan successful is a "better compress." This method of disk spindle compression differs from the recommended method in that it builds a second 3,000-block free space in addition to the one created by compressing the disk. The key to the better compress is to execute a BLDFILE command between alternate compresses of the spindle. (Figure 13-7 shows how this method can be applied to a two-spindle system.) To begin the better compress, execute the COMPRESS command to create free space at the end of the spindle that will contain high-activity objects. Then execute the BLDFILE command to create an empty 3,000-block file. The file retains free space after the second compress of the disk. Following a second compress of the spindle — this time in the opposite direction from the first compress — delete the empty file to create a second free space.

Figure 13-7

The better compress. (This is procedure STMBP01 on diskette.)

```
*  Compress A1 & A2 with 3,000 blocks of workspace near #LIBRARY
COMPRESS A1,FREELOW
BLDFILE ##SPACE,S,BLOCKS,3000,256,A1        WORK FILE TO BE DELETED
COMPRESS A1,FREEHIGH                        CREATE WORK SPACE
DELETE ##SPACE,F1
*
COMPRESS A2,FREEHIGH
**
BLDFILE ##SPACE,S,BLOCKS,3000,256,A2        WORK FILE TO BE DELETED
COMPRESS A2,FREELOW
DELETE ##SPACE,F1                           CREATE WORK SPACE
```

During the first few applications of their "better compress," Skowron and Langren moved objects to the appropriate "high-activity" and "low-activity" areas manually before they were satisfied they had the best organization possible. The moves were accomplished between the first and second compresses of the spindle after the empty file had been built. To begin the move, they used the CATALOG procedure (with LOCATION specified for the fifth parameter) to obtain a listing of block number locations and sizes for objects on each spindle. From this listing, they could determine a desirable order for the objects and pinpoint the changes needed in block number location (for more information about block number location for disks, see Chapter 4 of the *S/36 Concepts and Programmer's Guide.*

They then began to move objects according to these determinations. To move libraries, they used the ALOCLIBR command. Although designed to let users increase or decrease the size of a library, the ALOCLIBR command also can be used to change the location of a library. To change a library's location, Langren and Skowron specified a small increase or decrease (e.g., one block) in library size, specifying the new block number location for the disk preference parameter (parameter 4). To move folders, they used the MOVEFLDR command, indicating the block number preference in parameter 2.

Moving files, however, required a bit more planning because on the S/36 there is no ALLOCATE FILE or MOVE FILE command. The closest thing to either is the COPYDATA command. But because the COPYDATA command was created to duplicate data under a new file name (thus keeping the original data intact under the original file name), it does not let you remove a file from one location and place it in another. Therefore, Langren and Skowron used a three-step process for files. First, they renamed the original file using the RENAME command. Next, using the COPYDATA command, they copied the renamed version of the file to the new block number location (specified for parameter five) under the original file name. Finally, they deleted the renamed version of the file using the DELETE command. Moving index files also involved removing and rebuilding alternate indexes.

A Matter of Imbalance

After the initial manual reorganization of objects on the spindles, Langren and Skowron found that with continued use of the better compress, they did not need to rearrange objects on a particular disk very often. For the most part, the organization of objects on the spindles remained true to the original plan. However, they discovered that even with faithful use of the better compress, some imbalances (in the relative number of objects) between disk spindles occurred. Because unbalanced spindles can degrade system performance, the final problem that Langren and Skowron faced was devising a method of balancing the workload across all spindles to work with the better compress.

Imbalances among disk spindles in a multispindle system occur because the system always begins its search for space to allocate new objects on the least-used spindle when no spindle preference has been specified. Through an internal monitoring procedure, the system knows which spindles have had the most activity within the past operating hour. When a user creates a new object without specifying a spindle location, the system checks its spindle-activity figures and then attempts to allocate the object on the spindle with the lowest figure. In the case of a two-spindle system, the spindle with the least activity often is A2. (Because A1 contains all the system files, its activity level can remain higher throughout the operating day.) As a result, spindle A2 can become disproportionately filled with new objects.

This imbalance can degrade system performance because as A2 becomes filled, its disk seeks greater than one-third increase. So, one indication of spindle imbalance could be a disproportionate increase in one spindle's disk seeks. Another SMF counter that Langren and Skowron monitor to evaluate spindle balance is the Disk Utilization counter. This counter gives the percentage of activity for a spindle during the snapshot period. Near-equal percentages for all spindles indicate a balanced system. (The SMF counter is different from the internal monitor the system uses for deciding where to allocate objects.)

To rebalance spindles quickly, Langren and Skowron designed some automatic procedures for moving objects that they could use instead of the manual procedures deseribed earlier. One procedure uses the RENAME, COPYDATA, and DELETE commands to copy files from the high-activity area of spindle A2 to the high-activity area of spindle A1. Langren and Skowron included this procedure in their better compress after the initial compress of A2.

They designed another procedure, VDSKTOA3 (Figure 13-8), to move direet files, large index or sequential files, and PC files to a new spindle automatically, via a SAVE to and RESTORE from tape (or diskette). The procedure can be executed via the job queue if the file name and new spindle loeation are supplied. (Otherwise, the procedure prompts the operator for these parameters.) When the procedure is run, it initializes the tape, saves the file to tape, renames the file on disk, restores the original file to disk, and deletes the renamed version. A message tells the operator that the file has been moved to the desired spindle. This procedure is a little kinder to the system than the COPYDATA sequenee beeause it does not tie up two disk arms the way the COPYDATA sequence ean when it is used with large files.

Figure 13-8

Procedure VDSKTOA3

```
// IF JOBO-YES GOTO WORK
// * 'Moves PCDISK/DISK File via Tape-T1'
// * 'DO NOT USE THIS PROCEDURE IF FILE HAS ALTERNATIVE INOICES '
// * 'Mount scratch tape on T1, Volid-IBMIRO '
// * 'Enter the target drive location, A1, A2, or A3'
// LOCATION OFFSET-1,DATA-'?2R?'
// * 'Enter the name of the PCDISK/FILE to be moved
// JOBO 1,#LIBRARY,VDSKTOA3,?1R?,?2?
// RETURN
*
TAPEINIT T1,SL,IBMIRD,CLEAR
// WAIT INTERVAL-000010      REWIND TAPE
SAVE ?1?,,,IBMIRD,T1
// WAIT INTERVAL-000030     REWINO TAPE
RENAME ?1?,@@THE
RESTORE ?1?,,,,,?2?,T1
// IF DATAF1-?1? DELETE @@THE,F1
// MSG ?WS?  PCDISK/FILE '?1?' HAS BEEN MOVED TO ?2?
```

Extents of large, high-activity files also can thwart the disk organization associated with the better compress. Langren and Skowron discovered that the 3,000-block space was not always large enough to contain extents of their large, high-activity files; the extents were going to the larger free space on the opposite side of the disk. This tendency has two disadvantages: one, the disk seeks greater than one-third increase for that spindle; and two, the system "locks out" users from a file that is being extended — and because large files take a long time to extend, users could be locked out for a relatively long time.

To solve this glitch in their methodology and to improve file availability, Langren and Skowron designed a routine that would add more records to a specified file automatically each time it runs. The example routine shown in Figure 13-9 adds 1,000 records to the NEWSFILE file by evaluating parameter 11 as the length of the file plus 1,000 records and then running the COPYDATA sequence. Langren and Skowron ran this routine nightly when they compressed the disk spindles, thereby avoiding performance-degrading, large file extents during the work day. (Note: the COPYDATA statement in Figure 13-9 uses the REORG parameter to reorganize the index file in key-sequence order. This parameter is optional.)

Figure 13-9 *Procedure for increasing file size and reorganizing file. (This is procedure STMBP02 on diskette.)*

```
* Calculate new file size - file + 1000 records
// EVALUATE P10,7=?F'A,NEWSFILE'?
// EVALUATE P11=?10?+1000
*
RENAME NEWFILE,OLDNEWS
* 'REORG' parameter in COPYDATA is optional
COPYDATA OLDNEWS,,NEWSFILE,RECRODS,?11?,A3,,REORG
// IF DATAF1-NEWSFILE DELETE OLDNEWS,F1
```

As they became more comfortable with their disk management procedures, Langren and Skowron also began to practice some object-level management to help improve disk performance. Their goals were to reduce disk seeks greater than one-third and to keep active data as concise as possible. Their practices, listed below, can be used in any S/36 shop.

• Reorganize index files in key-sequence order, so when a file is read in key order, program buffers (or S/36 Cache) can be used efficiently and the disk arm does not need to move back and forth across the file to locate records.

• Process index files sequentially — when they are in key order — to avoid additional reads of the disk to access the index.

• Delete unused files from the spindle or move them to a low-activity area, so active data is kept together and the disk arm does not move over unused data during disk accesses.

• Purge unneeded records from master and history files. Again, unused data takes up space on the spindle and increases the likelihood of a disk seek greater than one-third occurring.

• Avoid over-allocation of files and libraries. Over-allocation creates unused space on the spindle and increases distances between active data.

• Keep source programs out of production libraries. Source programs are not needed in a production library (where object code is stored); they create a type of "unused space" in that location because the disk arm must move over them to find the next production object.

One automated procedure that Langren and Skowron used to recover unused space in folders was Jeff Silden's FLDCMP (Figures 13-10a and 13-10b). The FLDCMP procedure uses system utilities and a custom program to write and execute a CONDENSE command for each user-created DW/36 folder on the system. The procedure is equivalent to a "Condense All" for folders. (Extents and IBM-generated folders are not included.)

As they put these concepts into practice, Langren and Skowron continued to streamline and automate their disk management system. They ran the resulting routines nightly to keep their three-spindle system efficiently organized. By automating their disk-tuning routines, Langren and Skowron also reduced the time the system was dedicated during the work day for manual reorganization of the spindles. Any S/36 shop can and should adopt similar practices, based on the recognition that the disk is the slowest component of a S/36 and therefore offers the most performance gain when kept well-tuned.

To complete Langren and Skowron's recommendations for tuning your S/36, part 3 of "Managing S/36 Performance" will discuss system memory management and offer guidelines for setting S/36 Cache and organizing a nighttime job queue.

Figure 13-10a

Procedure FLDCMPP

```
**  Procedure name  -  F L D C M P P              Sept 11, 1986
*   Function - All DW/36 folders reorg'd to min. size & then bumped.
// * 'FLDCMPP is running. Optimizing your Displaywrite folders!'
// IF DATAF1-VTOC      DELETE VTOC,F1
CATALOG ALL,F1,,,NAME,,VTOC
// LOCAL OFFSET-201,DATA-'050'    * Amount to bump the folder for coming week
// LOAD FLDCMP
// FILE NAME-VTOC,RETAIN-S,DBLOCK-?F'A,VTOC'?
// FILE NAME-FLDRS,LABEL-?WS?.FLDRS,BLOCKS-3,EXTEND-2,DBLOCK-96
// RUN
// LOAD $MAINT
// FILE NAME-?WS?.FLDRS,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-?WS?.FLDRS,TO-#LIBRARY,RETAIN-R
// END
CMPFLDR ,     * PROC CREATED BY FLDCMP
**  End of Procedure - F L D C M P P
```

Figure 13-10b

Program FLDCMP

```
*..  .  1 ...  . 2 . . . . 3 . .  .. 4      . 5 .     . 6 . . . .. 7 .      8
0001 H/TITLE Creates Proc to Compress/Alloc all Folders     * FLDCMP *
0002 H       64                         B                                FLDCMP
0003 FVTOC     IP  F 132 132            DISK
0004 FFLDRS    O   F  80  80            DISK
0005 **  Program Name - F L D C M P               Sept 11, 1986
0006 **  Function - Generates OCL stream to compress all Displaywrite
0007 **    folders, and then re-allocate at LDA-defined blocks over the minimum.
0008 **  Operation - L3 break on name causes check if FLTYP is a folder
0009 **     If it is, we make sure it's not an DW-supplied one. If still
0010 **     ok, get the folder name right justified and generate OCL lines
0011 **     for a compressed save followed by ALOCFLDR
0012 E                    NAM         8  1
```

```
E                    FILD       8  1
** Input Specifications
I* VTOC is the direct-to-disk output of a CATALOG by name.
I*  the I-specs ONLY recognize records with an "F" in the "type" column
IVTOC     NS  01   26 CF
I                                        1   8 NAME  L3
I                                        1   1 NAM1
I                                       26  32 FLTYP
IVTOC     NS  02
** Local Data Area                          How much larger than minimum
I           UDS
I                                      201 2030BUMP
**   End of Input Specifications  /  Start of Calculations
C   91                 SETOF                     91    For LOAD/RUN
C   90                 GOTO $NRML                      Only for 1st cycle
C                      SETON                     9091
**   Start of "regular processing"
C           $NRML      TAG
C                      SETOF                     10    Every cycle
C                      MOVE *BLANK    FILD             Clear leftovers
**     L3 detail is 1st record for a new VTOC name. Folders can
**       have multiple entries.
C   L3      FLTYP      IFEQ 'FOLDER '                  Only do folders
C   L3      NAME       IFNE 'WPDOCS  '                 But not IBM's
C   L3      NAM1       IFNE '#'
C                      SETON                     10    Enable output
C                      MOVEANAME      NAM
C                      Z-ADD8         I         10
**    Note usage here—"I" is length 1. Subtracting I from 9 below
**     is appropriate. It gets the pointer bumped by 1 for the alignment
C           LOOP       TAG                             We want to get the
C           NAM,I      COMP *BLANKS                  32-library name left-
C   32                 SUB  1         I          31      justified. Scan
C   32 31              GOTO LOOP                         for rt-most char
C           9          SUB  I         I                Now, put into write
C                      MOVEANAME      FILD,I            array
**
C                      END
C                      END
C                      END
OFLDRS   D         91
O                                  23 '// COPY LIBRARY-P,NAME-'
O                                  39 'CMPFLDR,RETAIN-R'
O*  Do ALOCFLDR MIN allows a fragmented folder to get equiv of CONDENSE
OFLDRS   D         L3 10
O                                  18 'ALOCFLDR          ,'
O                        FILD      17
O                                  21 'MIN'
O*  Now do an ALOCFLDR INCR to up the amount of space.
OFLDRS   D         L3 10
O                                  18 'ALOCFLDR          ,'
O                        FILD      17
O                                  23 'INCR,'
O                        BUMP      26
OFLDRS   T         LR
O                                   7 '// CEND'
** End of Program - F L D C M P
```

Managing S/36 Performance - Part 3 Improving Performance by Merging Memory

by Debra Kahn

Code on diskette:
Procedures JOBQ1, JOBQ3, #SCHED1, #SCHED2
RPG programs JOBQ02, JOBQ03

Maximize the performance of your company's S/36 by tuning cache sizes, setting job queue priorities, and using night job queue procedures.

IBM supplies S/36 MIS managers with two resources for managing the system's internal memory: S/36 cache and the S/36 multistream job queue. You can control system throughput — and thus system performance — by adjusting cache values and setting job queue priorities based on your S/36 work load and job mix. In managing these two resources, Rebecca Langren and Bob Skowron, MIS staff for Duke Communications International (DCI), shared the goal of all MIS managers: increase system throughput to maximum efficiency while maintaining acceptable user response times. To that end, Langren and Skowron devised rules for managing cache and the multistream job queue that you can adapt to your own performance management plan.

Cache: Not Just Small Change

S/36 cache lets you control multiple disk accesses systemwide by sharing data already in memory. This sharing can translate into faster throughput times by lessening the effect of virtual paging on data access. Properly used, cache can be the cornerstone of any S/36 memory management plan. But, if improperly used or used with too little available memory, cache can compound an already high virtual paging rate with many writes to disk from cache pages.

Langren and Skowron were familiar with the benefits and drawbacks of implementing S/36 cache, but they needed criteria for deciding whether to implement the resource and when and how to modify it. IBM has spelled out some criteria, but those sources lack the comprehensive perspective Langren and Skowron needed to judge whether cache could help their system's performance and whether they had applied the resource effectively. So, they began to develop their own cache implementation criteria. The first step was to determine whether implementing cache would in fact help them meet their performance goal. To determine cache's usefulness on their system, Langren and Skowron applied two rules of thumb culled from reading and from talking with experts.

The first rule was that caching works best when memory is not a constraint and when the job mix includes consecutive processing of shared or unshared files or random processing of a heavily shared file. Because DCI's S/36 has plenty of memory and because much of the job mix centers on processing one large, heavily shared customer file, Langren and Skowron suspected that implementing S/36 cache would help them use their memory resource more efficiently.

The second rule of thumb was that System Measurement Facility (SMF) User Area Disk Activity (UADA) counts of between 75 and 125 per minute mean the system is already using its memory resources productively and that cache could help increase throughput. The UADA count reflects the amount of shuffling of system and user objects between main memory and disk (i.e., transient loads and virtual paging) when memory is overcommitted. In DCI's case, UADA counts were well within the acceptable range, and Langren and Skowron concluded that proper cache management could yield measurable performance improvement.

Planning for Cache Efficiency

After determining that implementing S/36 cache would indeed help meet their performance goal, Langren and Skowron collected baseline data by recording several weeks of SMF data and by noting completion times for weekly and daily jobs. Langren and Skowron then devised an implementation plan based on careful monitoring of cache-related SMF values.

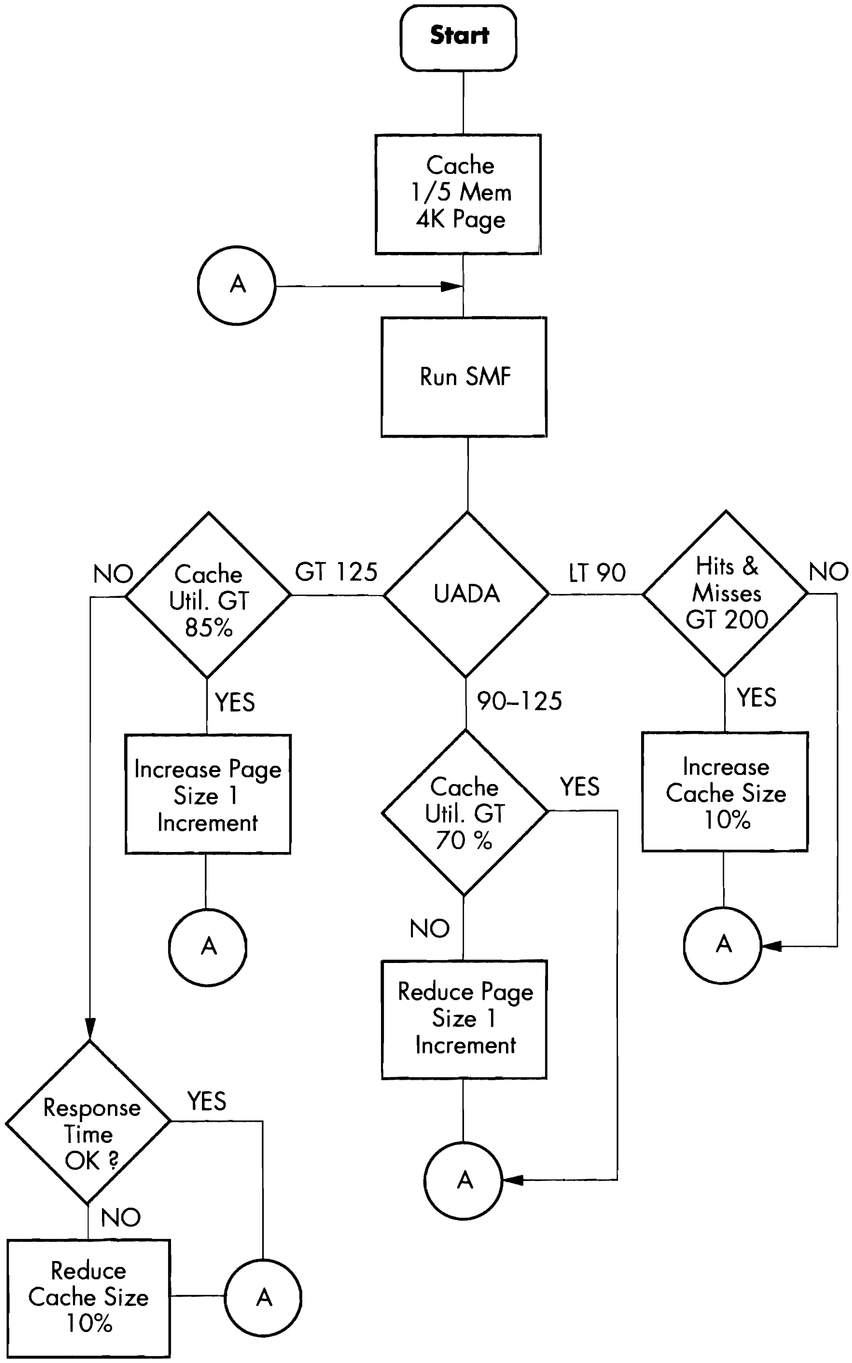
Effective cache management requires achieving optimum cache size by tuning two cache parameters: the amount of user area allocated to cache in main storage (at least 64 K) and the size of cache pages (1 K, 2 K, 4 K, 8 K, or 16 K). Furthermore, the ratio between these two numbers must allow the system to create at least 32 pages in cache. Langren and Skowron recognized that they had to monitor SMF reports to determine when to modify cache sizes and by how much. During previous planning efforts, Langren and Skowron had chosen two cache-related SMF counters to monitor in addition to UADA: cache utilization (i.e., the average and maximum percentage of cache reads found in memory) and cache hits and misses (i.e., the number of times the system found needed data in memory and the number of times the system did not). Minimum acceptable cache utilization is 40 percent, and the minimum ratio of cache hits to misses is 2:1.

Based on experience, Langren and Skowron then devised a decision flowchart. Their flowchart (Figure 13-11) is an effective performance management tool you can use to help tune cache on your S/36. The flowchart helps you analyze cache efficiency by letting you use SMF values to evaluate memory usage. The flowchart also provides paths to follow for fine-tuning cache when memory performance lags.

The flowchart's starting values for implementing cache are conservative: one-fifth of your system's total memory size (specified for parameter two of the CACHE command) and a 4 K page size (specified for parameter three). After you start cache with these values, you should monitor your SMF counters for a sufficient length of time (several hours during each of your different computing environments — e.g., interactive day work and batch night work) to determine whether the values fall within acceptable

Figure 13-11

The cache decision flowchart



boundaries. Be sure to monitor during peak times to get the best feel for how well your system performs with cache. You also should determine whether interactive response time is acceptable. To do this, you can use the S/36 RMF (Response Time Measurement Facility) or informally monitor user satisfaction through surveys and "walkthroughs."

If cache utilization, hit-to-miss ratio, and UADA fall within acceptable ranges (i.e., cache utilization of at least 40 percent, a hit-to-miss ratio of at least 2:1, and a UADA of 75 to 90 pages per minute), and if response time is good, cache is helping your system, and you should notice some improvement in job completion times compared to baseline times collected before you implemented cache. But if your UADA, for instance, is creeping out of the "comfort zone" of 75 to 90 pages per minute, you should follow the flowchart to fine-tune your cache values.

To help you make such adjustments, the flowchart contains three yes-no branches, each based on a different range of UADA values. Each branch asks you to examine your cache SMF values further and suggests you alter cache sizes according to the results of that examination. (Remember, any time you alter cache sizes, you still must meet the 32-page minimum, so adjusting one value may mean adjusting the other also.) Each branch returns you to the "run SMF" instruction because you must monitor system performance continuously to judge whether further changes in cache are necessary.

The key to adjusting cache is to keep in mind both your cache utilization, which reflects what portion of cache storage is being used, and your hit-to-miss ratio, which reflects how effectively cache is keeping needed data in memory. Adjusting cache page size can improve your cache utilization percentage. But again, consider your job mix. If your system usually processes files that contain large records, increasing cache page size lets the system place more records into each page, thus increasing the likelihood that the system will locate a needed record in that cache page and improve cache utilization. The same is not true, however, for files with small records. Figure 13-12 illustrates this principle.

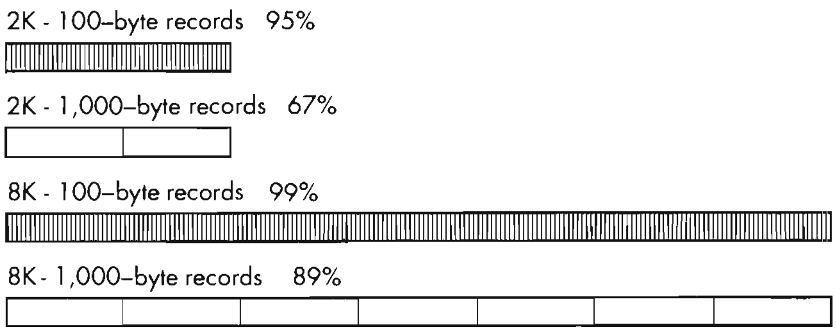


Figure 13-12
How records in a cache page affect utilization

Increasing cache page size from 2 K to 8 K when your system processes 100-byte records gains you only a 4 percent increase in cache utilization. But the same increase in page size gains you a 22 percent increase in cache utilization if your system processes 1,000-byte records.

Increasing cache size in addition to page size results in more records available in memory, thus increasing the likelihood of a cache hit and improving your hit-to-miss ratio. But too many cache pages can affect efficiency as much as too few. Because the system writes data from cache to disk as soon as the user program calls for a write operation, cache efficiency suffers when the system performs too many writes (and subsequent disk reads) in a short time to keep up with the demand for necessary data. Hence, the left and center branches of the flowchart instruct you to readjust page sizes to keep cache utilization within acceptable limits.

Also remember that you can alter cache sizes to match your job mix. Langren and Skowron found that increasing cache page sizes each evening improved their system's handling of the regularly scheduled evening batch work, which called for very large sequential reads of the customer file.

Working with Job Priorities

Langren and Skowron's efforts to increase their system's throughput also focused on job scheduling techniques that enhance the S/36's multistream work capacity. The separate responsibilities of the Main Storage Processor (MSP) and the Control Storage Processor (CSP) and the existence of virtual storage in addition to real address space in the user area of memory mean that the S/36 can work on many user jobs at once, both interactive and batch. Common sense, however, dictates that you schedule large batch jobs and low-priority jobs so they do not interfere with high-priority interactive work. In addition, every company experiences times when, either to meet a deadline or a managerial objective, the system must work on one job before all others. By letting you assign priorities to jobs, the S/36 job queue gives you the means to schedule jobs and therefore the means to control your system's work stream.

But S/36 operators, if they use the job queue at all, often don't know which priority they should assign to user jobs. Such uncertainty defeats the purpose of the job queue and can mean the system is struggling to accommodate more workload than it should. To understand how the S/36 job queue can help increase system throughput by controlling the work stream, let's review the basics of setting up the job queue and then examine two schemes that Langren and Skowron devised — and that you can use — to enhance performance.

To check the status and priority of any job in the job queue, you must examine the job queue status screen (Figure 13-13a). In addition to current status, this screen tells you how your job queue is set up. That is, it tells you the maximum number of jobs you've allowed to be run from the job queue ("Max Active Jobs") and how many jobs from each priority group can run at one time ("Max for PRTY").

Figure 13-13a
Job queue status screen

```
    Complete                    JOB QUEUE STATUS
    Jobs in Queue:   0 of  100         JOBQ PRTY  STOPPED  : 0
       Max Active Jobs: 3   Max for PRTY 5:1  4:2  3:1  2:1  1:1  0:1
                                                              PRIORITY
  POS  JOBNAME   PROC/DOC    LIBR/FLDR    USER      STATUS    JOBQ PROC

 SYS-5689 The job queue is empty now



   Cmd7-End      Cmd8-Help     Cmd15-Update      Cmd16-Restart      Roll-
Page
```

Figure 13-13b
JOBQUEUE command screen

```
                               JOBQUEUE
                        Jobs on the job queue
     1. Display specific job(s)           6. Start the job queue
     2. Put a job on the queue            7. Stop the job queue
     3. Cancel a job                      8. Change position of a job
     4. Hold a job                        9. Change processing priority
     5. Release a job                    10. Change maximum active jobs
     Ready for option number or command
```

Figure 13-13c
Job queue change maximum screen

```
                             CHANGE COMMAND                       Optional-*
                 Change maximum number of active job queue jobs

Job queue or job queue priority . . . . . . . JOBQ,0,1,2,3,4,5  JOBQ      *

Number of jobs . . . . . . . . . . . . . . . . . . . . . 1 - 50

Cmd3-Previous menu
```

Langren and Skowron allow no more than three jobs to be run from the job queue at once, and only one per priority. The three-job maximum contrasts starkly with the default 50-job maximum for the S/36 job queue, but Langren and Skowron found that allowing more than three jobs to run concurrently severely degrades user response times. Few, if any, shops that do interactive work should operate with a job queue set at a 50-job maximum.

To determine your maximum setting, experiment with different values and monitor how they affect user response times and SMF values. To change the maximum job total or the maximum number of jobs for a priority, you must summon the job queue command screen (Figure 13-13b) by typing HELP JOBQUEUE from the system console. Then take option 10 and complete the appropriate parameters (Figure 13-13c).

In addition to ensuring that user response times are not compromised by running too many jobs from the job queue at one time, Langren and Skowron wanted to ensure that scheduled jobs do not compete with one another or with emergency jobs. To meet this objective, they limited the number of jobs per priority to one and set up concrete criteria for each priority. Their priority list (Figure 13-14) reflects a sensible approach to scheduling emergency requests — giving them the highest priority of 5 — and considers the company's system requirements.

Figure 13-14
Example job queue priority assignments

Priority 5 (Highest):	Emergency requests
Priority 4:	Express work DW/36 letters Short batch jobs
Priority 3:	DMAS CMAS MAPICS Accounting applications Subscription fulfillment
Priority 2:	Large batch jobs DFU lists Data merge with DW/36 Query/36 jobs
Priority 1:	Application development and maintenance

Most of DCI's users occasionally print unique correspondence on company letterhead using DisplayWrite/36. Langren and Skowron gave such letter-printing jobs the next-highest priority (4). Langren and Skowron also used priority four for short batch jobs needed to keep other work flowing. Langren and Skowron placed work to be done by IBM applications (DCI uses only DMAS) at priority three to avoid changing IBM's recommended job queue priority. In addition, Langren and Skowron gave a priority of three to accounting applications and subscription fulfillment tasks because these jobs are essential to the company's day-to-day operation. At priority two, Langren and Skowron placed large batch jobs, DFU lists, data merges, and Query/36 work because these jobs require large amounts of resource and often are not crucial to daily operations. Finally, application development and application maintenance, which often has no specific deadline, are at the lowest priority.

You can decide whether to construct your own priority criteria or adapt Langren and Skowron's by considering your company's system requirements. As always, be sure to monitor SMF values and user satisfaction to ensure that your criteria are on the mark. In addition, you should secure support from management and users for your job queue scheme by explaining how your scheme furthers the company's business goals. Langren and Skowron took consideration of their company's system one step further and created a nighttime job queue and night work scheduler to accommodate additional user workload.

Night Owls Need Work

DCI's nighttime job queue consists of a set of procedures and programs Langren and Skowron include in their daily work to facilitate job scheduling. Procedure JOBQ1 (Figure 13-15a) and program JOBQ02 (Figure 13-15b) let users submit jobs to a "Night JOBQ" by placing those jobs' parameters in a special file during the day. Procedure JOBQ1 then evokes procedure JOBQ3 (Figure 13-15c) and program JOBQ03 (Figure 13-15d), which remain inactive until the nightly maintenance procedure completes its work, to read the file and then load and run the submitted jobs. By looking at the file's contents (using POP's file browse, for example), the system operator can easily review the list of jobs submitted to the nighttime job queue and make adjustments before leaving for the day.

Figure 13-15a
JOBQ1 procedure to load nighttime job queue

```
// LOCAL OFFSET-1,DATA-'?1?',BLANK-256
// LOCAL OFFSET-9,DATA-'?2?'
// LOCAL OFFSET-17,DATA-'?3?'
// LOCAL OFFSET-25,DATA-'?4?'
// LOCAL OFFSET-33,DATA-'?5?'
// LOCAL OFFSET-41,DATA-'?6?'
// LOCAL OFFSET-49,DATA-'?7?'
// LOCAL OFFSET-57,DATA-'?8?'
// LOCAL OFFSET-65,DATA-'?9?'
// LOCAL OFFSET-73,DATA-'?10?'
// LOCAL OFFSET-81,DATA-'?11?'
// LOCAL OFFSET-89,DATA-'?12?'
// LOCAL OFFSET-97,DATA-'?13?'
// LOCAL OFFSET-105,DATA-'?14?'
// LOCAL OFFSET-113,DATA-'?15?'
// LOCAL OFFSET-121,DATA-'?16?'
// LOCAL OFFSET-129,DATA-'?17?'
// LOCAL OFFSET-137,DATA-'?18?'
// LOCAL OFFSET-145,DATA-'?19?'
// LOCAL OFFSET-153,DATA-'?20?'
// LOCAL OFFSET-161,DATA-'?21?'
// LOCAL OFFSET-169,DATA-'?22?'
// LOCAL OFFSET-177,DATA-'?23?'
// LOCAL OFFSET-185,DATA-'?24?'
// LOCAL OFFSET-193,DATA-'?25?'
```

Figure 13-15b

Related program JOBQ02

```
// LOCAL OFFSET-201,DATA-'?26?'
// LOCAL OFFSET-209,DATA-'?27?'
// LOCAL OFFSET-217,DATA-'?28?'
// LOCAL OFFSET-225,DATA-'?29?'
// LOCAL OFFSET-233,DATA-'?30?'
// LOCAL OFFSET-241,DATA-'?31?'
**
** Keep workstn ID to notify user when task has completed in proc-JOBQ3
// LOCAL OFFSET-249,DATA-'?WS?'
** If you do not install the programs in #LIBRARY add the library name
** LOAD JOBQ02,library
// LOAD JOBQ02
// FILE NAME-IN,LABEL-INPRC,
// IF DATAF1-INPRC EXTEND-1
// ELSE DISP-NEW,BLOCKS-4
// RUN
** CHG To RETURN--will start JOBQ3 after nightly backup
// PAUSE 'This job has been submitted to the Night JOBQ '
// RETURN
** If you do not install the programs in #LIBRARY add the library name
// IFF ACTIVE-JOBQ3 EVOKE JOBQ3
** IFF ACTIVE-JOBQ3,library EVOKE JOBQ3,library
```

Figure 13-15c

JOBQ3 procedure to run nighttime job

```
*         1         2 .       3       . 4         5         6         7         8
H/TITLE  UPDATE FILE INPRC WITH NITEQ PARMS
H     024                                   1                              JOBQ02
FIN      O   F 256 256           DISK                           A
ILDADS       UDS
I                                           1 256 LDATA
C                       MOVELLDATA     RECORD256
C                       SETON                       LR
OIN      D        LR
O                           RECORD   256
// IFF DATAF1-INPRC RETURN
// TAG $START
** if you do not install the programs in #LIBRARY add the library name
** LOAD JOBQ03,library
// LOAD JOBQ03
// FILE NAME-IN,LABEL-INPRC
// RUN
// EVALUATE P1=?L'1,8'?  P2=?L'9,8'?  P3=?L'17,8'?  P4=?L'25,8'?  +
            P5=?L'33,8'?  P6=?L'41,8'?
// EVALUATE P7=?L'49,8'?  P8=?L'57,8'?  P9=?L'65,8'?  P10=?L'73,8'?  +
            P11=?L'81,8'?  P12=?L'89,8'?
// EVALUATE P13=?L'97,8'?  P14=?L'105,8'?  P15=?L'113,8'?  P16=?L'121,8'?  +
            P17=?L'129,8'?
// EVALUATE P18=?L'137,8'? +
            P19=?L'145,8'? +
            P20=?L'153,8'? +
            P21=?L'161,8'? +
            P22=?L'169,8'? +
            P23=?L'177,8'? +
            P24=?L'185,8'? +
            P25=?L'193,8'? +
            P26=?L'201,8'? +
            P27=?L'209,8'? +
            P28=?L'217,8'? +
            P29=?L'225,8'? +
            P30=?L'233,8'? +
            P31=?L'241,8'? +
            P32=?L'249,2'?
// LIBRARY NAME-?1?
// ?2? ?3?,?4?,?5?,?6?,?7?,?8?,?9?,?10?,?11?,?12?,?13?,?14?,?15?,?16?,+
   ?17?,?18?,?19?,?20?,?21?,?22?,?23?,?24?,?25?,?26?,?27?,?28?,?29?,?30?
**
** Send msg to submitter workstn to let user know work successfully completed
// LIBRARY NAME-?CLIB?
// MSG ?32?,PROCEDURE ?2? HAS COMPLETED ON THE NIGHT JOBQ '
// IF DATAF1-INPRC GOTO $START
// RETURN
```

Figure 13-15d
Related program JOBQ03

```
*  .    1   .    2         3         4         5         6         7         8
0001 H/TITLE  SET UP PREP TO RUN ACTIVE JOBS & TAG ONCE RUN IN FILE INPRC
0002 H     034                                1                           JOBQ03
0003 FIN      UP  F 256 256            DISK
0004 IIN      NS  01   1NC~
0005 I                                          1 256 RECORD
0004 I        NS  02
0006 ILDADS      UDS
0007 I                                          1 256 LDATA
0008 I                                          1   8 LIBR
0009 I                                          9  16 PROC
0010 I                                         17  24 PARM1
0011 I                                         25  32 PARM2
0012 C                         MOVE *BLANKS   LDATA
0013 C   01                    MOVELRECORD    LDATA
0014 C   01                    SETON                     LR
0015 C*
0016 C  NO1                    MOVEL'#LIBRARY'LIBR
0017 C  NO1                    MOVEL'DELETE  'PROC
0018 C  NO1                    MOVEL'INPRC   'PARM1
0019 C  NO1                    MOVEL'F1      'PARM2
0020 OIN      D        01
0021 O                                      1 '~'
```

Shell-scheduling procedures help Langren and Skowron schedule jobs they need to run only on certain days of the week. Procedure #SCHED1 (Figure 13-16a) simply records the day of the week by loading an LDA value and then evokes the procedure #SCHED2 (Figure 13-16b), which waits until 10 p.m. before commencing. Langren and Skowron use #SCHED2 to load all daily jobs that can run at night without operator assistance. After running the daily jobs, the procedure can evaluate the day-of-the-week value and run the appropriate jobs for that day.

Figure 13-16a
#SCHED1 procedure to evoke night work

```
// IF SWITCH8-1 GOTO SKIP
// IF ?2?/X * 'Invalid date  Please answer the question again''
// * 'What day is it? (Enter   MO,TU,WE,TH,FR,SA OR SU)'
// IF ?1R?/  RESET #SCHED1   ,X
// IF ?1?/MO EVALUATE P1-6
// IF ?1?/TU EVALUATE P1-5
// IF ?1?/WE EVALUATE P1-4
// IF ?1?/TH EVALUATE P1-3
// IF ?1?/FR EVALUATE P1-2
// IF ?1?/SA EVALUATE P1-1
// IF ?1?/SU EVALUATE P1-7
// IFF ?1?>0 RESET #SCHED1    ,X
// IF ?1?>7 RESET #SCHED1    ,X
// EVALUATE P2-' '
// TAG SKIP
// EVOKE #SCHED2 ?1?
// RETURN
*******************************************************************
# AUTOMATIC SCHEDULE PROGRAM TO RUN DAILY WORK AT NIGHT WITHOUT
# OPERATOR ASSISTANCE NEEDED  OPERATOR WILL NEED TO LET SYSTEM
# KNOW WHAT DAY IT IS  THIS PROC CALLS #SCHED2
*******************************************************************
```

Figure 13-16b
#SCHED2 procedure to run night work

```
// REGION SIZE-2
// SWITCH XXXXXXX1
// WAIT TIME-220000                        (10:00 P.M.)
// REGION SIZE-64
// EVALUATE P1-?1?-1
****************** DAILY JOBS BELOW
**
****************** END DAILY JOBS
```

```
********************
******************** WEEKLY JOBS BELOW
// IF ?1?/5 MON010                                  *MONDAY ONLY
// IF ?1?/4 TUE010                                  *TUESDAY ONLY
// IF ?1?/3 WED010                                  *WEDNESDAY ONLY
// IF ?1?/2 THU010                                  *THURSDAY ONLY
// IF ?1?/1 FRI010                                  *FRIDAY ONLY
// IF ?1?/0 SAT010                                  *SATURDAY ONLY
// IF ?1?/0 EVALUATE P1-7                           *SATURDAY ONLY
******************** END WEEKLY JOBS
// RESET #SCHED1 ?1?
*******************************************************************
# PROCEDURE THAT IS PART OF THE AUTOMATIC DAILY RUN SYSTEM  THIS
# PROCEDURE ACTUALLY CHECKS WHICH DAY WAS SELECTED BY THE OPERATOR
# EARLIER AND THEN RUNS THE PROCEDURE AT  SET TIME
*******************************************************************
```

You can adapt the night queue and scheduling procedures to your shop by using the appropriate tasks and values. Be sure all tasks in the nighttime job queue and all daily night work can stand truly "operatorless" operation. That is, you must anticipate necessary operator responses and error messages with the appropriate procedural language. Scheduling night jobs to increase overall system throughput is helpful only when they are completed successfully!

Good scheduling devices and appropriate job queue priorities can help you feed the system its tasks more efficiently and can increase overall MIS efficiency. But to these external devices, a good MIS manager always compares performance with organization requirements to understand how the system supports its users and the company goals. Only from this vantage point can you experiment with performance-tuning techniques.

Evaluating Cache Performance with SMF

by Ron Elliott

program by Sven Johnson

Code on diskette:
Procedure SMFP21
RPG programs SMFP21, SMFP23

Use this handy utility to bring all the cache-critical values from consecutive SMF snapshots together into one report.

Performance is an issue that affects all computer installations. One of the most powerful weapons that S/36 shops can employ in the battle for better performance is cache. Using cache is pointless, however, unless you measure results and tailor the cache parameters of buffer size and page size to your system. The System Measurement Facility (SMF) is a powerful tool that gathers statistics on system performance, but the voluminous report it produces makes it difficult to narrow in on the numbers relevant for cache analysis. Utility SMFP21 solves that problem by selecting important information from SMF and presenting it in an easy-to-use, condensed format.

The Essentials of Cache and SMF

If you are not familiar with the usage of either cache or SMF, you have some homework to do before you can benefit from utility SMFP21. Briefly, cache is a S/36 function that lets you allocate part of your main storage as a high-speed input/output buffer area that is shared systemwide. To balance the benefits of cache against possible disadvantages, you must choose the right values for cache page size and buffer size. The bigger the cache buffer, the more likely requested disk records are to be in the buffer; but if the cache area is too big, system performance suffers for lack of main storage. And if the total cache size or page size is too small, the time spent moving pages of data into the buffer from disk will negate the advantage of fast buffer retrieval.

SMF is an IBM-supplied utility that takes snapshots of system activity at user-specified intervals. The utility contains four separate procedures: SMF-START to begin collection of snapshot data, SMFSTOP to end data collection, SMFPRINT to print an SMF report, and SMFDATA to create a report data file. The average SMF detail report is many pages long, making it difficult to locate the cache storage size, page size, utilization, and UADA (user area disk activity) data necessary to analyze your use of cache. Although SMF offers a summary report, that report doesn't reflect changes you've made to cache storage size and page size and still contains nonpertinent information.

Care and Feeding of Utility SMFP21

Before you can use utility SMFP21, you must key in procedure SMFP21 (Figure 13-17), program SMFP21 (Figure 13-18), and program SMFP23 (Figure 13-19) and also prepare input data while running cache. You collect data using SMF procedures SMFSTART and SMFSTOP and later convert the collected data to the proper input format by running procedure SMFDATA.

Depending on your desired results, the time periods and snapshot intervals you choose for running SMF data collection (SMFSTART and SMFSTOP) and the parameters you choose for cache can vary. If your goal is to improve system efficiency during periods of peak activity, use a snapshot interval of one to three minutes, and collect SMF data at times of peak system activity every day for up to 10 days. (Data from separate days accumulates in default file SMF.LOG or your named file as long as you continue to use the same file name.) If you want to see how different cache sizes and page sizes affect performance, use the CACHE ALTER command to vary those values while SMF is active. For example, you can begin with a small amount of storage allocated to cache and increase it a little every 10 minutes, varying the page size at the same time. If you want to tailor your use of cache to different activity levels during the day, run SMF for an entire work day while still systematically changing cache parameters. If you have previously established standard cache values for your system, you may want to collect only a brief period of peak activity for review purposes.

When you have enough SMF data, run procedure SMFDATA (use report option ALL and default report file SMF.DATA) to convert the collected data to the SMF.DATA format expected by program SMFP21.

With procedure and program files in place and SMF.DATA at ready, run procedure SMFP21 to create the cache analysis report (see Figure 13-20). This report lists the performance statistics for each snapshot interval on a single line. The last values on each line of data — Disk Cache Storage (DCS) and Cache Page Size (DCP) — are the cache parameters in use at the time of the snapshot.

What Does It All Mean?

Here are some quick tips on using the report:

• The Main Storage Processor (MSP) value should be less than 60 percent.

• The Control Storage Processor (CSP) value should be less than 65 percent.

• Use of cache should decrease the percentage of time that each disk is used (shown under report headings A1, A2, A3, and A4). With or without cache, you should try to balance disk use evenly across the spindles on your system.

• If User Area Disk Activity (UADA), which is the sum of Translated Transfer Loads (TTL) plus Swaps-in (SWI) plus Swaps-out (SWO), is greater than 300 per minute, your system is suffering from lack of main storage (you may have allocated too much to cache). Keeping UADA below 200 swaps per minute is preferred.

• The Storage Releases figures reflect how often programs with relatively high priorities had to release storage. Any non-zero numbers in these columns are another indicator of lack of main storage.

• Disk Cache Hits (DCH) and Disk Cache Misses (DCM) indicate how often records sought by program reads were found or not found in the cache area. The ratio of hits to misses should be at least 2:1.

• The Disk Cache Utilization (DCU) column is the percentage of the disk cache read operations found in the disk cache. You should strive to make this value average 80 to 90 percent — and you never want it below 60 percent.

The SMFP21 summary report gathers together all the critical SMF values so that you can see how caching or various combinations of cache page and buffer sizes affect the MSP, CSP, disk usage values, swapping, and storage releases. With the "at a glance" analysis this report provides, you can find your optimum cache values easily.

Figure 13-17

Procedure SMFP21

```
*****************************************************
**     CAP DIAGNOSIS S/36       CAP GEMINI BRA/SJ
**
**     ANALYSIS WHEN CHANGING CACHE
**
**     SELECT DATA FROM SMF-DATA  PROGRAM SMFP21
**     PRINT REPORT               PROGRAM SMFP23
*****************************************************
*
// IF DATAF1-Y.FIL1 DELETE Y.FIL1,F1
*
// LOAD SMFP21
// FILE NAME-SMFDATA,LABEL-?1'SMF.DATA'?
// FILE NAME-SMFFIL1,LABEL-Y.FIL1,RECORDS-5000,EXTEND-5000
// RUN
*
// LOAD SMFP23
// FILE NAME-SMFFIL2,LABEL-Y.FIL1,RETAIN-S
// RUN
```

Figure 13-18

Program SMFP21

```
*        1         2         3         4         5         6         7         8
     H      04    Y                                                          SMFP21
     F*************************************************************
     F*          CAP DIAGNOSIS S/36        CAP GEMINI BRA/SJ
     F*
     F*          PROGRAM NAME    SMFP21
     F*          INPUT.          SMFDATA   SMF STANDARD REPORT FILE
     F*          OUTPUT          SMFFIL1   SELECTED DATA CONCERNING
     F*                                    MSP, CSP, DISKS, UADA, CACHE ETC
     F*
     FSMFDATA IP  F 800  80            DISK
     FSMFFIL1 O   F 800 100            DISK
     I*
     ISMFDATA NS  01   1 CA   2 CA   3 CA
     I*       IPL CONFIGURATION RECORD
     I                                        1   3 SMFRCD
     I                                       22  27 SMFDAT
     I*       DEVICE USAGE RECORD ACA
     I        NS  01   1 CA   2 CC   3 CA
     I                                        1   3 SMFRCD
     I                                        4  12 STIME
     I                                       19  21 MSP
     I                                       22  24 CSP
     I                                       37  39 A1
     I                                       40  42 A2
     I                                       43  45 A3
     I                                       46  48 A4
     I        NS  01   1 CA   2 CC   3 CB
     I*       DEVICE USAGE RATES RECORD B
     I                                        1   3 SMFRCD
     I                                       25  27 DCU
     I                                       28  32 DCS
     I                                       33  35 DCP
     I        NS  01   1 CA   2 CK   3 CA
     I*       SYSTEM EVENT AND I/O COUNTERS RECORD A
     I                                        1   3 SMFRCD
     I                                       24  28 TTL
     I                                       34  38 SWI
     I                                       39  43 SWO
     I                                       44  48 SWF
     I        NS  01   1 CA   2 CK   3 CC
     I*       SYSTEM EVENT AND I/O COUNTERS RECORD C
     I                                        1   3 SMFRCD
     I                                       54  58 L30
     I                                       59  63 L3W
     I                                       64  68 L40
     I                                       69  73 L4W
     I        NS  01   1 CA   2 CK   3 CD
     I*       SYSTEM EVENT AND I/O COUNTERS RECORD D
     I                                        1   3 SMFRCD
     I                                        9  13 DCH
     I                                       14  18 DCM
```

```
I        NS  01
I                                          1   3 SMFRCD
I*
C          SMFRCD    IFEQ 'AAA'
C                    MOVE SMFDAT    SMFDAX  6
C                    END
C*
C          SMFRCD    IFEQ 'ACA'
C                    MOVE STIME     UTIME   9
C                    MOVE MSP       UMSP    3
C                    MOVE CSP       UCSP    3
C                    MOVE A1        UA1     3
C                    MOVE A2        UA2     3
C                    MOVE A3        UA3     3
C                    MOVE A4        UA4     3
C                    END
C*
C          SMFRCD    IFEQ 'ACB'
C                    MOVE DCU       UDCU    3
C                    MOVE DCS       UDCS    5
C                    MOVE DCP       UDCP    3
C                    END
C*
C          SMFRCD    IFEQ 'AKA'
C                    MOVE TTL       UTTL    5
C                    MOVE SWI       USWI    5
C                    MOVE SWO       USWO    5
C                    MOVE SWF       USWF    5
C                    END
C*
C          SMFRCD    IFEQ 'AKC'
C                    MOVE L30       UL30    5
C                    MOVE L3W       UL3W    5
C                    MOVE L40       UL40    5
C                    MOVE L4W       UL4W    5
C                    END
C*
C          SMFRCD    IFEQ 'AKD'
C                    MOVE DCH       UDCH    5
C                    MOVE DCM       UDCM    5
C                    EXCPTOUTREC
C                    END
C*
OSMFFIL1 E               OUTREC
O                        UTIME  B   9
O                        UMSP   B  12
O                        UCSP   B  15
O                        UA1    B  18
O                        UA2    B  21
O                        UA3    B  24
O                        UA4    B  27
O                        UTTL   B  32
O                        USWI   B  37
O                        USWO   B  42
O                        USWF   B  47
O                        UL30   B  52
O                        UL3W   B  57
O                        UL40   B  62
O                        UL4W   B  67
O                        UDCH   B  72
O                        UDCM   B  77
O                        UDCU   B  80
O                        UDCS   B  85
O                        UDCP   B  88
O                        SMFDAX    94
```

Figure 13-19

Program SMFP23

```
*    . 1 . . .   2 . .    3 . .   . 4   . . . 5 . . .  . 6  .   . 7 . . . . . . 8
H      04    Y                                                          SMFP23
F***************************************************************
F*          CAP DIAGNOSIS S/36       CAP GEMINI BRA/SJ
F*
F*          PROGRAM NAME:  SMFP23
F*          INPUT:         SMFFIL2   SELECTED DATA FROM PROGRAM SMFP21
F*          OUTPUT:        LISTA     REPORT CAP DIAGNOS REPORT NR 4
```

```
F*                                   ANALYSIS WHEN CHANGING CACHE
F*
F*          INDICATORS     01        RECORD INDICATOR
F*                         66        WRITE HEADERS WITH SMF-LOG DATE
F*                         L1        WRITE TOTAL AND AVERAGE FIGURES
F*
FSMFFIL2 IP   1000 100           DISK
FLISTA   O         132     OF    PRINTER
ISMFFIL2 NS  01
I                                       1   60TIME
I                                      10  120MSP
I                                      13  150CSP
I                                      16  180A1
I                                      19  210A2
I                                      22  240A3
I                                      25  270A4
I                                      28  320TTC
I                                      33  370SWI
I                                      38  420SWO
I                                      43  470SWF
I                                      48  520L30
I                                      53  570L3W
I                                      58  620L40
I                                      63  670L4W
I                                      68  720DCH
I                                      73  770DCM
I                                      78  800DCU
I                                      81  850DCS
I                                      86  880DCP
I                                      89  940SMFDATL1
C*
C* TO GET HEADRS WITH SMF-LOG DATE
C*
C   OF               SETOF                         66
C   66               GOTO CONT
C                    EXCPTHDR
C                    SETON                         66
C          CONT      TAG
C*
C* ADD TO GET TOTALS
C*
C          MSP       ADD  MSPX      MSPX    60
C          CSP       ADD  CSPX      CSPX    60
C          A1        ADD  A1X       A1X     60
C          A2        ADD  A2X       A2X     60
C          A3        ADD  A3X       A3X     60
C          A4        ADD  A4X       A4X     60
C          TTC       ADD  TTCX      TTCX    60
C          SWI       ADD  SWIX      SWIX    60
C          SWO       ADD  SWOX      SWOX    60
C          SWF       ADD  SWFX      SWFX    60
C          L30       ADD  L30X      L30X    60
C          L3W       ADD  L3WX      L3WX    60
C          L40       ADD  L40X      L40X    60
C          L4W       ADD  L4WX      L4WX    60
C          DCH       ADD  DCHX      DCHX    80
C          DCM       ADD  DCMX      DCMX    80
C          DCU       ADD  DCUX      DCUX    60
C          DCS       ADD  DCSX      DCSX    60
C          DCP       ADD  DCPX      DCPX    60
C          X         ADD  1         X       30
C*
C* DIVIDE TOTALS TO GET AVERAGE FIGURES
C*
CL1        MSPX      DIV  X         MSPY    30H
CL1        CSPX      DIV  X         CSPY    30H
CL1        A1X       DIV  X         A1Y     30H
CL1        A2X       DIV  X         A2Y     30H
CL1        A3X       DIV  X         A3Y     30H
CL1        A4X       DIV  X         A4Y     30H
CL1        TTCX      DIV  X         TTCY    60H
CL1        SWIX      DIV  X         SWIY    60H
CL1        SWOX      DIV  X         SWOY    60H
CL1        SWFX      DIV  X         SWFY    60H
CL1        L30X      DIV  X         L30Y    60H
CL1        L3WX      DIV  X         L3WY    60H
```

```
CL1       L40X      DIV  X        L40Y    60H
CL1       L4WX      DIV  X        L4WY    60H
CL1       DCHX      DIV  X        DCHY    80H
CL1       DCMX      DIV  X        DCMY    80H
CL1       DCUX      DIV  X        DCUY    30H
CL1       DCSX      DIV  X        DCSY    40H
CL1       DCPX      DIV  X        DCPY    20H
CL1                 SETON                    OF
OLISTA    E  204         HDR
O                                  20 'C A P   G '
O                                  30 'E M I N I '
O                                  40 '  D I A G '
O                                  50 'N O S   S '
O                                  60 '/ 3 6     '
O                                  70 'REPORT DAT'
O                                  71 'E'
O                        UDATE     85 '  -  -  '
O         E  1           HDR
O                                  20 'ANALYSIS W'
O                                  30 'HEN CHANGI'
O                                  40 'NG CACHE  '
O                                  50 '   REPORT '
O                                  60 'NR 4      '
O                                  70 'SMF-LOG  D'
O                                  75 'ATE  '
O                        SMFDAT    85 '  -  -  '
O         E  1           HDR
O                                  10 'MSP - MAIN'
O                                  20 ' STOR PROC'
O                                  30 '  CSP - CO'
O                                  40 'NTR STOR P'
O                                  50 'ROC  TTL -'
O                                  60 ' TRANSF LO'
O                                  70 'ADS    CAC'
O                                  80 'HE DCH - H'
O                                  90 'ITS  DCM -'
O                                 100 ' MISSES  D'
O                                 110 'CU - UTILI'
O                                 120 'ZED       '
O         E  2           HDR
O                                  10 'SWI - SWAP'
O                                  20 'S IN  SWO '
O                                  30 '- SWAPS OU'
O                                  40 'T  SWF - S'
O                                  50 'WAPS FORCE'
O                                  60 'D         '
O                                  80 '   DCS - C'
O                                  90 'ACHE SIZE '
O                                 100 'DCP - CACH'
O                                 110 'E PAGE SIZ'
O                                 120 'E         '
O         E  1           HDR
O                                  10 '          '
O                                  20 '          '
O                                  30 ' —  DISK-'
O                                  40 '——      '
O                                  50 '—— UAD'
O                                  60 'A ———'
O                                  70 '—     —S'
O                                  80 'TORAGE REL'
O                                  90 'EASES—  '
O                                 100 '    ——  '
O                                 110 '— CACHE -'
O                                 120 '———    '
O         E  2           HDR
O                                  10 '      TIME '
O                                  20 '  MSP CSP '
O                                  30 '    A1  A2 '
O                                  40 ' A3  A4    '
O                                  50 '  TTL    SW'
O                                  60 'I   SWO    '
O                                  70 'SWF      L3'
O                                  80 'O   L3W    '
O                                  90 'L40    L4W '
O                                 100 '      DCH  '
O                                 110 '    DCM DCU'
```

```
O                                  120 '  DCS DCP '
O         D        N1P
O                            TIME   B    9 '       .    '
O                            MSP   3B   15
O                            CSP   3B   19
O                            A1    3B   25
O                            A2    3B   29
O                            A3    3B   33
O                            A4    3B   37
O                            TTC   3B   45
O                            SWI   3B   51
O                            SWO   3B   57
O                            SWF   3B   63
O                            L3O   3B   71
O                            L3W   3B   77
O                            L4O   3B   83
O                            L4W   3B   89
O                            DCH   3B   98
O                            DCM   3B  106
O                            DCU   3B  110
O                            DCS   3B  115
O                            DCP   3B  119
O         T 22      L1
O                                    9 'TOTAL     '
O                            TTCX  3B   45
O                            SWIX  3B   51
O                            SWOX  3B   57
O                            SWFX  3B   63
O                            L3OX  3B   71
O                            L3WX  3B   77
O                            L4OX  3B   83
O                            L4WX  3B   89
O                            DCHX  3B   98
O                            DCMX  3B  106
O         T  2      L1
O                                    9 'AVERAGE  '
O                            MSPY  3B   15
O                            CSPY  3B   19
O                            A1Y   3B   25
O                            A2Y   3B   29
O                            A3Y   3B   33
O                            A4Y   3B   37
O                            TTCY  3B   45
O                            SWIY  3B   51
O                            SWOY  3B   57
O                            SWFY  3B   63
O                            L3OY  3B   71
O                            L3WY  3B   77
O                            L4OY  3B   83
O                            L4WY  3B   89
O                            DCHY  3B   98
O                            DCMY  3B  106
O                            DCUY  3B  110
O                            DCSY  3B  115
O                            DCPY  3B  119
```

Continued

Figure 13-20

Sample cache analysis report

C A P   G E M I N I   D I A G N O S   S / 3 6   REPORT DATE   8-16-89

ANALYSIS WHEN CHANGING CACHE   REPORT NR 4   SMF-LOG DATE   89-08-04
MSP - MAIN STOR PROC   CSP - CONTR STOR PROC   TTL - TRANSF LOADS   CACHE DCH - HITS   DCM - MISSES   DCU - UTILIZED
SWI - SWAPS IN   SWO - SWAPS OUT   SWF - SWAPS FORCED   DCS - CACHE SIZE DCP - CACHE PAGE SIZE

DISK——	—— UADA ———					—STORAGE RELEASES—				—— CACHE ——									
TIME	MSP	CSP	A1	A2	A3	A4	TTL	SWI	SWO	SWF	L30	L3W	L40	L4W	DCH	DCM	DCU	DCS	DCP
15.23.17	12	24	14	9	3	0	71	43	12	0	0	0	0	0	355	97	79	1200	8
15.25.17	37	44	37	22	14	0	152	46	29	0	0	0	0	0	2187	543	80	1200	8
15.27.17	20	41	20	10	66	0	71	17	0	0	0	0	0	0	586	310	65	1000	4
15.29.18	27	41	8	15	80	0	56	25	1	0	0	0	0	0	1283	442	74	1000	4
15.31 18	16	23	4	8	9	0	31	14	1	0	0	0	0	0	610	319	66	1000	4
15.33.18	24	31	11	14	10	0	78	19	21	0	0	0	0	0	1305	349	79	1000	4
15.35.18	27	30	7	11	16	0	31	27	5	0	0	0	0	0	1478	576	72	1000	4
15.37.18	8	16	6	6	7	0	40	11	1	0	0	0	0	0	336	159	68	1000	4
15.39.19	18	25	18	8	9	0	60	52	14	0	0	0	0	0	730	341	68	1000	4
15.41 19	26	44	19	16	92	0	57	24	11	0	0	0	0	0	1181	610	66	1000	4
15.43.19	44	55	44	16	67	0	195	89	37	0	0	0	0	0	1035	366	74	1000	4
15.45.20	60	49	11	12	51	0	40	28	14	0	0	0	0	0	1197	330	78	1000	4
15.47.20	56	55	12	6	53	0	74	36	22	0	0	0	0	0	878	376	70	1000	4
15.49.20	71	49	21	12	27	0	60	44	8	0	0	0	0	0	866	383	69	1000	4
15.51.21	47	46	29	32	18	0	120	33	9	0	0	0	0	0	1469	802	65	1000	4
15.53.21	20	32	31	36	4	0	93	33	11	0	0	0	0	0	1452	534	73	1000	4
15.55.21	39	50	46	29	18	0	94	42	9	0	0	0	0	0	2103	935	69	1000	4
15.57.22	20	46	9	11	86	0	60	29	0	0	0	0	0	0	635	326	66	1000	4
15.59.22	31	48	13	18	60	0	81	43	13	0	0	0	0	0	1376	682	67	1000	4
16.01.22	17	40	8	8	65	0	86	24	1	0	0	0	0	0	449	153	75	1000	4
16.03.22	27	44	9	13	92	0	57	28	10	0	0	0	0	0	1246	714	64	800	2
16.05.23	20	33	11	8	59	0	51	11	0	0	0	0	0	0	341	287	54	800	2
16.07.23	19	37	5	7	98	0	30	15	0	0	0	0	0	0	692	451	61	800	2
16.09.23	13	35	7	8	69	0	42	26	2	0	0	0	0	0	318	102	76	800	2
16.11.23	22	40	6	9	81	0	28	11	0	0	0	0	0	0	891	621	59	800	2
16.13.24	12	16	5	44	4	0	29	25	9	0	0	0	0	0	1845	371	83	1400	16
16.15.24	21	29	27	15	3	0	214	21	6	0	0	0	0	0	2237	151	94	1400	16
16.17.24	14	25	62	15	3	0	160	18	0	0	0	0	0	0	1448	122	92	1400	16
16.19.25	10	22	62	12	3	0	106	7	0	0	0	0	0	0	1317	80	94	1400	16
16.21.25	10	21	55	14	2	0	125	13	0	0	0	0	0	0	1223	93	93	1400	16
16.23.26	16	28	78	16	3	0	173	18	1	0	0	0	0	0	1555	138	92	1400	16
16.25.27	14	28	20	14	3	0	91	17	0	0	0	0	0	0	1577	225	88	1400	16
16.27.27	9	22	11	9	5	0	87	9	2	0	0	0	0	C	1429	199	88	1400	16
16.29.27	7	18	12	7	4	0	72	13	3	0	0	0	0	0	865	129	87	1200	8
16.31.27	7	11	3	3	4	0	11	9	3	0	0	0	0	0	259	48	84	1200	8
16.33.27	12	13	8	4	2	0	50	18	18	0	0	0	0	0	287	62	82	1200	8
16.35.28	4	10	8	1	1	0	24	43	6	0	0	0	0	0	190	28	87	1200	8
16.37.28	6	14	7	1	2	0	35	44	11	0	0	0	0	0	98	37	73	1200	8
16.39.28	5	10	4	1	1	0	29	7	7	0	0	0	0	0	168	27	86	1200	8
16.41.28	13	24	6	1	11	0	30	8	1	0	0	0	0	0	234	344	40	1200	8
16.43.28	7	16	5	1	3	0	29	26	3	0	0	0	0	0	131	67	66	1200	8
16.45.29	4	10	3	1	1	0	19	5	0	0	0	0	0	0	93	21	82	1200	8
16.47.29	1	2	1	0	0	0	5	4	0	0	0	0	0	0	29	6	83	1200	8
16.49.29	3	6	4	1	1	0	26	9	0	0	0	0	0	0	65	23	74	1200	8

O A P   G E M I N I   D I A G N O S   S / 3 6   REPORT DATE   8-16-89

ANALYSIS WHEN CHANGING CACHE   REPORT NR 4   SMF-LOG DATE   89-08-04
MSP - MAIN STOR PROC   CSP - CONTR STOR PROC   TTL - TRANSF LOADS   CACHE DCH - HITS   DCM - MISSES   DCU - UTILIZED
SWI - SWAPS IN   SWO - SWAPS OUT   SWF - SWAPS FORCED   DCS - CACHE SIZE DCP - CACHE PAGE SIZE

DISK——	—— UADA ———					—STORAGE RELEASES—				—— CACHE ——									
TIME	MSP	CSP	A1	A2	A3	A4	TTL	SWI	SWO	SWF	L30	L3W	L40	L4W	DCH	DCM	DCU	DCS	DCP
16.59.30	8	29	9	3	81	0	16	27	0	0	0	0	0	0	114	10	92	1200	8
17.01.30	8	16	34	3	29	0	3	2	0	0	0	0	0	0	315	468	40	1200	8
17.02.46	4	15	20	2	22	0	25	10	0	0	0	0	0	0	83	35	70	1200	8
TOTAL							3230	1210	313	0	0	0	0	0	41424	13808			
AVERAGE	25	28	17	10	30	0	56	21	5	0	0	0	0	0	714	238	77	1131	7

Monitoring Realtime Memory Usage

by Gary T. Kratzer

program by Mel Beckman

Code on diskette:
Procedure MMETER
RPG program MMETER
Assembler subroutine SUBR$S
Screen format member MMETERFM

Utility MMETER helps you monitor S/36 realtime memory use to improve performance.

Understanding your S/36's memory helps you manage it more effectively. This knowledge is far more useful if you can monitor it, keeping a close watch on how memory use affects system performance. It is helpful to know, for example, how program swapping affects memory use. Although the S/34 reflects swapped programs on the STATUS USERS (D U) display, the S/36 often swaps only certain pages of a program, which makes reflecting swaps on the D U display impractical. Nor does IBM supply S/36 swapping information through a utility. Virtual page use, additional information that helps you monitor memory, also is unavailable through an IBM-supplied utility.

But the MMETER utility lifts the curtain that conceals your S/36's memory use. MMETER gives you a *realtime* account of how nucleus pages, user main and sub programs, system programs, and system workspaces are using S/36 memory.

MMETER's realtime memory account helps you track down intermittent memory-related performance problems that are hard to catch through the System Measurement Facility (SMF) report you already may use to monitor memory. For example, if you experience occasional drastic increases in response time at unpredictable intervals of days, or even weeks, you may not be able to establish a useful performance measurement with SMF.

To check your system's "normal" memory use, use utility MMETER when system response time is good. You then can compare memory use during good response times to memory use during slow response times, easily determining whether memory overcommitment is a possible source of trouble.

To use the MMETER utility, simply key MMETER to display the S/36 Memory Meter screen shown in Figure 13-21. The information on the MMETER screen reflects how memory in your system currently is being used. To update the information shown, press Enter. To end MMETER, press Command key 7.

The MMETER utility comprises RPG program MMETER (Figure 13-22), screen format member MMETERFM (Figure 13-23), assembler subroutine SUBR$S, and procedure MMETER (Figure 13-24). (SUBR$S, used by program MMETER as an RPG SPECIAL file, gathers current memory information to provide realtime analysis of memory use each time you press Enter from the MMETER screen.)

Figure 13-24
Sample MMETER display

```
                    System/36 Memory Meter

                                 Total        Currently
                       Count   committed      paged in
Nucleus pages            74     148K   29%     148K   28%
User main programs..     12     508K   99%      92K   17%
System programs. ..     195     448K   88%      60K   12%
User sub programs        40   1,280K  250%     204K   40%
System workspaces  .      7      54K   11%       8K    2%

TOTALS  .  ..   . ..          2,438K  476%     512K  100%
   (unused memory)..                             0K
   (main storage size)                         512K
```

MMETER Headings

At the top of the MMETER screen are three column headings, described below.

Count is the number of programs, pages, or workspaces for the line item.

Total committed refers to the total kilobytes and percentage of virtual (committed) storage for each line item.

Currently paged in refers to the kilobytes and percentage of real (main) memory currently used by each line item. Note that the total committed memory (2,438 K, or 476 percent, in the example shown in Figure 13-24) considerably exceeds the amount of real memory physically installed on the machine — a prime example of the S/36's virtual memory management scheme at work.

MMETER Line Items

Utility MMETER displays information for five line items — nucleus pages, user main programs, system programs, user sub programs, and system workspaces — that tell you how and where your system is using memory.

Nucleus pages consist of the fixed and variable nucleus areas, which are used by the system and always reside in real (not virtual) storage. The amount of memory that nucleus pages consume changes as the system gives and takes pages to and from the user area.

User main programs consist of all user application programs and SSP utilities (such as $MAINT, $COPY, compilers).

System programs are SSP programs called by other system programs to perform repetitive tasks. They run *transparently* to the user (e.g., spool writers, the initiator, the command processor, system transients) and, as a result, are excluded from the D U display.

User sub programs are of interest only to users of external program call facilities products that let you call other RPG programs (e.g., ASNA's RPG/III or BPS's RPG 2 1/2). Generally, "stock" S/36 application programs do not have sub programs, so the memory used by sub programs is not available through any IBM-supplied utility.

System workspaces show the pages of memory the system uses for storing various tables (such as the active procedure list and active screen formats) and program buffers (such as those created when a program exceeds its 64 K address space and must place file buffers into the task work area).

MMETER's individual line items help you isolate the memory requirement for user programs, called programs, or system activity. A high memory overcommitment for either the *User main programs* or *User sub programs* is due to the application workload directly under your control — reducing the workload will help even out response time peaks. Excessive overcommitment in the *System workspaces* line item usually results from heavy use of IBM Office Products like DisplayWrite/36 and Personal Services/36; these programs require a large amount of virtual memory for document manipulation. To improve performance, you must add more memory or reduce Office Product use for nonpeak hours.

Overcommitment of memory to either the *Nucleus pages* or *System programs* line items usually is caused by high SSP activity, either through a large volume of procedure interpretation or a large number of medium-lived System Queue space (SQS) items. If you don't add memory to relieve the high memory requirement, you probably will need to modify your programs to reduce their dependency on SQS or procedure execution.

MMETER Totals

The *Totals* for the line items show the cumulative kilobytes and percentage of committed memory and the kilobytes and percentage of real memory currently being used by the system. As previously stated, the amount committed can be many times the amount being used.

Unused Memory is often zero kilobytes, but having no unused memory is not necessarily a cause for concern. It usually means that your installed memory is being used to its fullest potential, thereby providing maximum benefits in throughput and response times.

However, if the total kilobytes and percentage of committed memory is consistently high, you might want to consider adding more memory to your machine. For example, if the normal total memory commitment for your system is 230 percent, but it increases to 500 percent during slow response times, purchasing additional memory probably will help alleviate the problem.

Using the MMETER utility regularly to monitor memory use gives you the inside scoop on whether and where memory constraints are contributing to sagging response times. Because utility MMETER is interactive, you can invoke it at a moment's notice, and unlike SMF, MMETER lets you visually compare *realtime* memory use with *current* system activities.

Figure 13-22

Program MMETER

```
*   1         2         3 . . . . 4 .       5   .     6         7 .       8
H    028                       B       1                             MMETER
F* System/36 Memory Meter, by Mel Beckman
FSYSINFO IP        256            SPECIAL       SUBR$S
FWORKSTN CD  F     512            WORKSTN
IWORKSTN
ISYSINFO
I                                    B   1   20INV                   Invalid storage
I                                    B   3   40UPUSED                User programs used
I                                    B   5   60UPOWND                User programs owned
I                                    B   7   80SPUSED                System programs used
I                                    B   9  100SPOWND                System programs owned
I                                    B  11  120SYSTEM                System nucleus
I                                    B  13  140FREE                  Unused storage
I                                    B  15  160SWUSED                System workspaces used
I                                    B  17  180SWOWND                System workspaces owne
I                                    B  19  200TWUSED                Task workspaces used
I                                    B  21  220TWOWND                Task workspaces owned
I                                    B  23  240UPCNT                 User program count
I                                    B  25  260SPCNT                 System program count
I                                    B  27  280SWCNT                 System workspace count
I                                    B  29  300TWCNT                 Task workspace count
C* Compute page counts for invalid, free and system pages
C           INV       DIV  2         INVCNT  40
C           FREE      DIV  2         FRECNT  40
C           SYSTEM    DIV  2         SYSCNT  40
C* Compute amount of storage actually used
C           INV       ADD  SYSTEM    TOTU    40
C                     ADD  UPOWND    TOTU
C                     ADD  SPOWND    TOTU
C                     ADD  SWOWND    TOTU
C                     ADD  TWOWND    TOTU
C* Compute total storage commitment
C           INV       ADD  SYSTEM    TOTC    40
C                     ADD  UPUSED    TOTC
C                     ADD  SPUSED    TOTC
C                     ADD  SWUSED    TOTC
C                     ADD  TWUSED    TOTC
C* Compute main storage size
C           TOTU      ADD  FREE      MSS     40
C                     MOVELMSS       MSSO    50
C* Compute ratios as percentage of main storage size
C           SYSTEM    DIV  MSSO      SQSPCT  43       Nucleus
C           TOTU      DIV  MSSO      TOTUPC  43       Total used
C           TOTC      DIV  MSSO      TOTCPC  43       Total commitment
C           UPUSED    DIV  MSSO      UPUPCT  43       User prog used
C           UPOWND    DIV  MSSO      UPOPCT  43       User prog owned
C           SPUSED    DIV  MSSO      SPUPCT  43       Sys prog used
C           SPOWND    DIV  MSSO      SPOPCT  43       Sys prog owned
C           SWUSED    DIV  MSSO      SWUPCT  43       Sys workspace used
C           SWOWND    DIV  MSSO      SWOPCT  43       Sys workspace owned
C           TWUSED    DIV  MSSO      TWUPCT  43       Task workspace used
C           TWOWND    DIV  MSSO      TWOPCT  43       Task workspace owned
C           FREE      DIV  MSSO      FREPCT  43       Unused storage
C* Display results
C                     EXCPTSCREEN
```

```
0056 C                    READ WORKSTN                LRLR
0057 C    KG              SETON                     LR
0058 OWORKSTN E                SCREEN
0059 O                                  K8 'MMETER01'
0060 O                                  30 '    K     %      K     %'
0061 O                         SYSCNTZ   4
0062 O                         SYSTEMZ  10
0063 O                         SQSPCTZ  16
0064 O                         SYSTEMZ  23
0065 O                         SQSPCTZ  29
0066 O                                  60 '    K     %      K     %'
0067 O                         UPCNT Z  34
0068 O                         UPUSEDZ  40
0069 O                         UPUPCTZ  46
0070 O                         UPOWNDZ  53
0071 O                         UPOPCTZ  59
0072 O                                  90 '    K     %      K     %'
0073 O                         SPCNT Z  64
0074 O                         SPUSEDZ  70
0075 O                         SPUPCTZ  76
0076 O                         SPOWNDZ  83
0077 O                         SPOPCTZ  89
0078 O                                 120 '    K     %      K     %'
0079 O                         TWCNT Z  94
0080 O                         TWUSEDZ 100
0081 O                         TWUPCTZ 106
0082 O                         TWOWNDZ 113
0083 O                         TWOPCTZ 119
0084 O                                 150 '    K     %      K     %'
0085 O                         SWCNT Z 124
0086 O                         SWUSEDZ 130
0087 O                         SWUPCTZ 136
0088 O                         SWOWNDZ 143
0089 O                         SWOPCTZ 149
0090 O                                 180 '    K     %      K     %'
0091 O                         TOTC  Z 160
0092 O                         TOTCPCZ 166
0093 O                         TOTU  Z 173
0094 O                         TOTUPCZ 179
0095 O                                 210 'K      %'
0096 O                         FREE    203 '   0'
0097 O                         FREPCT  209 '   0'
0098 O                                 234 'K'
0099 O                         MSS   Z 233
```

Figure 13-23

Screen format member MMETERFM

```
*... ... 1 ... ... 2 ..   .. 3 ... ..  4 ... ... 5 ... ... 6 ... . . 7 .   . . 8
     SMMETER01           YY                                 G
     DFL0001    22 118Y                               CSystem/36 Memory Meter
     DFA0001     5 434Y                               CTotal
     DFA0002     9 445Y                               CCurrently
     DFL0002     5 524Y                         Y     CCount
     DFL0003    11 531Y                         Y     C committed
     DFL0004    11 544Y                         Y     C paged in
     DFL0005    22 6 2Y                               CNucleus pages. ... .
     DFL0006    30 625Y
     DFL0007    22 7 2Y                               CUser programs.     .
     DFL0008    30 725Y
     DFL0009    22 8 2Y                               CSystem programs....
     DFL0010    30 825Y
     DFL0011    22 9 2Y                               CTask workspaces. .
     DFL0012    30 925Y
     DFL0013    2210 2Y                               CSystem workspaces.  ..
     DFL0014    301025Y
     DFL0019    2212 2Y                               CTOTALS.  . .. .  ...
     DFL0020    301225Y
     DFA0001    2213 2Y                               C   (unused memory)
     DFL0016    301325Y
     DFA0002    2214 2Y                               C   (main storage size)
     DFA0001    301425Y
```

Figure 13-24

Procedure MMETER

```
* System/36 Memory Meter, by Mel Beckman
// LOAD MMETER
// RUN
```

Re-creating Subroutine SUBR$S

If you don't have assembler subroutine SUBR$S, you can re-create it with procedure MKSUBR$S (you don't need IBM's Assembler Language Program Product to install SUBR$S). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBR$S. You need to run MKSUBR$S only once because SUBR$S is subsequently linked into program MMETER when it is compiled.

```
// * 'Re-creating R-module SUBR$S in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D95BE2404000000040000000000000FF400000009900022000000000089'
// LOCAL OFFSET-273,DATA-+
'0330013231000000801300000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBR$S member to insert object code
// LOAD $FEFIX
// RUN
HDR  3BAA SUBR$00000
PTF  CE78 RSUBR$S,89,,#RPGLIB
DATA 0C7F 00 0000 E20BE2E4C2D95BE200000000C500000000000000000000000000000000000000
DATA 9443 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA D853 00 0040 E3360038340B00BF340100873402008B8C400D8D100EF08100AC010C1EF880BD
DATA 1A81 00 0060 B5020CAF3F3F3F35A100C4F4000FB02B008E010815F4000F8A2C0D0026080703
DATA CA8F 00 0080 E337008E38A100C4F281BF7D0004F201049E010115700104F20123700034F281
DATA 808C 00 00A0 109E0107139E0109158E011900C1F287009E0103139E0106159E011700002903
DATA 6106 00 00C0 E33700A800C1700304F201049E0100157D06D4F28213707F04F284009E010F13
DATA A07D 00 00E0 9E0111158E011B00C17DB004F282009E0113139E0115158E011D00C100372401
DATA CAD8 00 0100 E31000C475A107F18776F88000AE151515C2A10000C2A20000C087000000D100
DATA 56A3 00 0120 0000000000000000000000000000000000000000000000000000000000000000
DATA A3B1 00 0140 C50000C400000000000000000000000000000000000000000000000000000000
DATA 4E79 00 0180 0000000000000000000000000000000000000000000000000000000000000000
DATA 03CF 00 0180 615C000000000000000000000000000000000000000000000000000000000000
DATA D59A 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 0862 00 01C0 4040404040404040404040404040404040404040404040404040404040404040
DATA 4629 00 01E0 4040404040404040404040404040404040404040404040404040404040404040
END  F780
```

POP

CHAPTER

14

Retrieving Library and Member Information in POP

by Gary T. Kratzer

program by Chuck Lundgren

Code on diskette:
Procedure LIBRI
RPG program LIBRI
Screen format member LIBRIFM

Utility LIBRI is a POP enhancement for libraries and library members that gives you the speed and convenience of the file information opcode.

I will make a bold guess that about 90 percent of S/36 shops with a programming staff or consultant have IBM's POP utility. POP is a much-used programmer's tool for several reasons: it provides a full-screen editor and the capability to quickly display and browse multiple files, libraries, and diskette files, and it lets the programmer enhance POP by adding commands that can be called from the multiple display screens. Among POP's useful features are the one-character commands that you enter next to a desired item on a display screen to execute a variety of functions on files, libraries, and diskettes.

One such POP opcode is the FILE utility's I (information) opcode. FILEI is far easier and faster than running a CATALOG when you need file information. This opcode displays detailed information about a file such as its type, number of records used, record length, file key information, and disk address, and your information is displayed instantaneously at the workstation rather than placed on the print queue. Unfortunately, there is no equivalent "I" function for libraries or library members — until now.

Utility LIBRI is a POP enhancement that is essentially a FILEI for library information. Normally, you must use the slow, inconvenient LISTLIBR procedure to retrieve library information and print or place it in a file. But by adding LIBRI to your supply of POP opcodes, you can instantly get the library information you need.

Installing LIBRI in #POPLIB

Programmers familiar with POP know that it comes with a LIBRI function that calls the SAVELIBR procedure. If you plan to use our LIBRI, we recommend you rename the existing POP LIBRI procedure (to LIBRQ, possibly) because our naming convention is consistent with POP's FILEI operation — the I denotes "information." (If you decide not to rename the existing LIBRI, you must instead change the component names of our utility to LIBR*x* — *x* being the letter you choose for the opcode.)

Utility LIBRI consists of RPG program LIBRI (Figure 14-1), screen format member LIBRIFM (Figure 14-2), and procedure LIBRI (Figure 14-3). Note that program LIBRI uses assembler subroutine SUBRLD. To install LIBRI, simply copy program and procedure LIBR and screen format member LIBRIFM into #POPLIB or #LIBRARY.

Using LIBRI

With utility LIBRI installed, you can place an I next to any library or library member on the POP libraries display to retrieve the detailed information shown on the screens in Figures 14-4a, 14-4b, and 14-4c. When you select a library, the Library Information screen in Figure 14-4a appears. This screen includes not only the library size and disk location, but also the total number of each type of library member (i.e., object, subroutine, procedure, and source). This information is not available on a LISTLIBR report. In addition, the screen displays the number of diskettes required to save the library in each of the four S/36 diskette formats.

From the Library Information screen, you have three choices: press Enter for the same screen for the next library you have chosen; enter a new library name and press Enter; or press Command key 7 to return to the POP libraries display. If you enter a library name that doesn't exist, the Library Error screen in Figure 14-5a gives you the option to enter a valid library name or to press Command key 7 to return to the POP libraries display.

If you need information for an individual library member, place an I next to that member's name on POP's library members display. If you choose a source or procedure member, you see the screen in Figure 14-4b; select an object or subroutine, and you get the screen in Figure 14-4c. These screens are similar, but directory entry fields (such as "Link edit" and "RLD displacement") that don't apply to source members or procedures do not appear on the screen in Figure 14-4b. Both Member Information screens display the member's numeric subtype along with its literal description (e.g., subtype 35 is RPG, subtype 40 is "unspecified"). The screens also display member attributes literally (e.g., SUNGLOW Program), rather than the list of 1s and 0s in a LISTLIBR detail report. As in the case of the previous screens, you may press Enter to display the next member chosen, enter a new member type or library name and press Enter, or press Command key 7 to return to the POP libraries display. If you enter an invalid member or library name, the screen in Figure 14-5b lets you correct the error.

How LIBRI Works

POP's three main programs, FILE, LIBR, and DKET, basically work the same way. They display up to 64 objects on a screen and let you operate on those objects either by entering a one-letter opcode next to any object or by pressing a command key. When you enter an opcode or press a command key, these three programs decide whether they should execute the operation internally or have an external procedure execute it. Internal operations are executed first, followed by external procedures.

The FILE, LIBR, and DKET programs execute several opcodes and command keys internally, such as FILEI or the B opcodes that let you browse a file, library member, or diskette object. No matter which opcodes you enter on the display screen, all internal opcodes are processed as a group with all intervening operations deferred. For some internal operations, such as copying multiple members, POP groups together all like operations and displays them on a confirmation screen — which is why you can copy up to 64 members at once.

For the externally executed commands, POP uses a clever naming scheme. Each command corresponds to a procedure with the name of the command formatted *aaaax* for opcodes or *aaaa*KY*nn* for command keys (where *aaaa* is FILE, LIBR, or DKET, *x* is the opcode, and *nn* is the command key number). If you are looking at a file list and you press Command key 15, POP would, hypothetically, execute procedure FILEKY15 in #POPLIB. If you enter an E next to a library member, POP executes the LIBRE procedure, which runs POP's full-screen editor program.

For externally executed commands, POP queues up to 12 commands for execution. This command list consists of 12 10-byte elements stored in the LDA from location 51 through 170. For example, if you enter print command P, restore command J, and delete command D for three libraries, POP executes the delete command internally and puts the print and restore commands in a list (Figure 14-6) to execute the following procedures:

```
LIBRP LIBTEST,L,0
LIBRJ #IDALIB,L,0
```

If you decide to use utility LIBRI on a system without POP, you must write a procedure to store the correct values in the LDA, beginning with position 51.

Emulating FILEI with LIBRI

LIBRI emulates the internally executed POP FILEI command (which, like other internal operations, runs before all external commands) by executing all library I operations together. LIBRI executes the first I opcode (subroutine GETNX in Figure 14-1) and scans the command list for subsequent I codes (subroutine GETPC), checks for errors (subroutines CHKLB and CHKMR), executes the I opcodes (subroutine GETIN), and removes them by compressing the command list (subroutine FIRST). Given the command list in Figure 14-7, for example, LIBRI shows the information for libraries #IDALIB and FSLIB and then compresses the list as shown in Figure 14-8 to prevent the I opcodes from being re-executed when the LIBR# procedure regains control. Thus, LIBRI has the same "look and feel" as FILEI and is also very efficient because program LIBRI does not have to be reloaded several times to process several enqueued requests.

If, after using POP, you wondered how you ever got along without it, I'm sure you'll find yourself thinking the same thing again after trying utility LIBRI. It's that convenient, efficient, and natural to use.

Figure 14-1

Program LIBRI

```
*   1         2         3         4         5         6         7         8
H     064                    B                                     LIBRI
F*************************************************************************
F* PROGRAM NAME      LIBRI
F* DESCRIPTION       POP-like information screen for libraries and
F*   library members  Modeled after POP's "I" opcode in the FILE
F*   display
F* PROGRAMMER        Chuck Lundgren (Iris Software, Inc.)
F*   (c) COPYRIGHT 1989 Iris Software, Inc  - All Rights Reserved
F* DATE WRITTEN      June 1989
F*
F*************************************************************************
FWORKSTN CD  F     520            WORKSTN
F                                             KFMTS  LIBRIFM
E*************************************************************************
E                    PCL        12 10                POP command list
E                    NCL        12 10                New POP command list
E                    MTYP    4   4  1                Member type list
E                    MCNT        4  4 0              Member totals for lib
E                    STA     1  34  2   STD     28   Sub-type descriptions
E                    AT      1  40 38                Attribute descriptions
E                    AL         11 38                Attribute screen list
E                    A8          5  1                Attribute byte array
E                    B           8  1                Bit array
I*************************************************************************
I* Library name input
I*
IWDRKSTN         9 CO
I                                       1   8 INPLI8
I*
I* Library member input
I*
I               18 C1
I                                       1   8 INPLI8
I                                       9  16 INPMBR
I                                      17  17 INPTYP
I*-----------------------------------
I* Library directory data structure
I*
IMEMDS       DS
I                                      10  15 DRADDR
I                                      16  180DR#TXT
I                                      19  22 DRLINK
I                                      23  270DR#STM
I                                      28  31 DRSCA
I                                      32  33 DRRLD
I                                      34  360DRCORE
I                                      37  37 DRATR1
I                                      38  38 DRATR2
I                                      39  39 DRATR3
I                                      40  410DRMRTN
I                                      40  41 DRMRTC
I                                      42  430DRREL
I                                      44  460DRTOTL
I                                      47  47 DRATR4
I                                      48  530DRMOD
I                                      54  590DRDATE
I                                      60  630DRTIME
I                                      64  65 DRATR5
I                                      66  69 DRPTF@
I                                      70  70 DRATR6
I*-----------------------------------
I* Library VTOC data structure
I*
ILI8DS       DS
I                                       1   6 L8FMT1
I                                       7  110L8L8SZ
I                                      12  150L8DRSZ
I                                      16  210L8USEC
I                                      22  270LBASEC
```

```
I                                         28  320LBUDIR
I                                         33  370LBADIR
I                                         38  43 LBBLIB
I                                         44  49 LBELIB
I                                         50  55 LBBDIR
I                                         56  61 LBEDIR
I                                         62  67 LBBMEM
I                                         68  73 LBEMEM
I                                         74  79 LBNMEM
I                                         80  80 LBENXT
I*------------------------------------
I* Single element from POP command list
I*
I            DS
I                                          1  10 PELEM                 Command list element
I                                          1   1 OPCOD                 POP operation code
I                                          2   9 POPLIB                Library name
I                                          2   9 POPMBR                Member name
I                                         10  10 POPTYP                Object type
I*------------------------------------
I* LDA containing the POP command list
I*
I           UDS
I                                          1   8 MBRLIB                Member's library name
I                                         51 170 PCL                   POP's command list
C/EJECT
C*************************************************************************
C* Main routine
C*
C                     EXSR FIRST                      Do first stuff
C*
C           END       DOUEQ'Y'                        Do until end of job
C                     EXSR GETNX                        Do next command
C                     END                             End DO
C*
C                     MOVEANCL       PCL              New POP cmd list
C                     SETON                     LR    End of job
C*
C*------------------------------------
C* Do first time processing
C*
C           FIRST     BEGSR
C*
C* Find location in POP command list containing this LIBRI.
C*
C           1         DO   12        LP      20       Do up to 12 times
C                     MOVE PCL,LP    PELEM              Fetch list elem.
C           OPCOD     IFEQ 'I'                          If it's "I",
C                     Z-ADDLP        FIRSTI  20           save location,
C                     Z-ADD12        LP                   and stop loop.
C                     END                               End IF
C                     END                             End DO
C*
C* Prevent subsequent LIBRI calls by removing them from POP's cmd. list.
C*
C           FIRSTI    ADD  1         LP               Next element.
C                     Z-ADDLP        NP      20       Point to new list
C           LP        DOWLE12                         For each command,
C                     MOVE PCL,LP    PELEM              Fetch list elem.
C           OPCOD     IFNE 'I'                          If isn't "I",
C                     MOVE PELEM     NCL,NP               put in new list
C                     ADD  1         NP                   next new elem.
C                     ELSE                              Else
C           OPCOD     IFEQ ' '                            If ' ' command
C                     MOVEL'0       'NCL,NP                 terminate w/0
C                     END                                 End IF
C                     END                               End IF
C                     ADD  1         LP                 next list elem
C                     END                             End DO
C*
C           NP        IFLE 12                         If in range
C                     MOVEL'0       'NCL,NP             terminate w/0
C                     END                             End IF
C*
C* Establish the initial command list position
C*
```

```
C                    Z-ADDFIRSTI    LP                  Point to first
C*
C*  Initialize the bit compare array
C*
C                    BITOF'01234567'X00     1           Set to X'00'
C                    MOVE X00       B                   Clear bit array
C                    BITON'0'       B,1                 Set up bit
C                    BITON'1'       B,2                   compare array
C                    BITON'2'       B,3
C                    BITON'3'       B,4
C                    BITON'4'       B,5
C                    BITON'5'       B,6
C                    BITON'6'       B,7
C                    BITON'7'       B,8
C*
C                    ENDSR
C*
C*-----------------------------------
C* Check for input errors library screen  If found, re-input
C*
C          CHKLB     BEGSR
C*
C          ERROR     DOUEQ*BLANK                        Do until no errors
C*
C                    EXIT SUBRLD                          Check lib exists
C                    RLABL          INPLIB                 Library name
C                    RLABL          INPMBR                 Always "*LIBR"
C                    RLABL          INPTYP                 Always "L"
C                    RLABL          LIBDS                  Output
C                    RLABL          RCODE  1               Output
C*
C          RCODE     IFEQ '1'                             If illegal libr,
C                    MOVE 'Y'       ERROR  1                flag error,
C                    EXCPTLIBERR                            library prompt,
C                    READ WORKSTN                 3030      read screen,
C  KG                MOVE *BLANK    ERROR                   If KG, clear er
C  KG                MOVE 'Y'       END                      & set eoj
C                    ELSE                                 Else no error
C                    EXSR SAVIN                             Save new inf
C                    MOVE *BLANK    ERROR                   Clear errorflag
C                    END                                  End IF
C*
C                    END                                End DO
C*
C                    ENDSR
C*
C*-----------------------------------
C* Check for input errors in member screen  If found, re-input
C*
C          CHKMR     BEGSR
C*
C          ERROR     DOUEQ*BLANK                        Do until no errors
C*
C                    EXIT SUBRLD                          Check mem exists
C                    RLABL          INPLIB                 Library name
C                    RLABL          INPMBR                 Member name
C                    RLABL          INPTYP                 Type
C                    RLABL          MEMDS                  Directory fields
C                    RLABL          RCODE                  Return code
C*
C                    SETON                      404142    Reset scrn inds
C          RCODE     COMP '1'                   4040      Illegal library?
C          RCODE     COMP '2'                   4141      Illegal member?
C          INPTYP    LOKUPMTYP                    11      Check obj type
C N11                SETOF                        42        Flag if error
C*
C          RCODE     IFGE '1'                             If errors
C                    MOVE 'Y'       ERROR                   Flag error,
C                    EXCPTMEMERR                            Show mem error
C                    READ WORKSTN                 3030      Read screen
C  KG                MOVE *BLANK    ERROR                   If KG, clear er
C  KG                MOVE 'Y'       END                      & set eoj
C                    ELSE                                 Else no error
C                    EXSR SAVIN                             Save new inf
C                    MOVE *BLANK    ERROR                   Clear errorflag
C                    END                                  End IF
```

```
C*
C                     END                                End DO
C*
C           CHKMRX    ENDSR
C*
C*-----------------------------------
C* Display and read screen
C*
C           DSPSC     BEGSR
C*
C           REQST     IFEQ 'LIBRONLY'                    If libr request
C                     EXCPTLIB                             display library
C                     ELSE                               Else
C           SAVTYP    COMP 'O'                       11    O module?
C  N11      SAVTYP    COMP 'R'                       11    R module?
C   11                EXCPTMEMOR                           Y-display it.
C  N11                EXCPTMEMSP                           N-disp  P or S.
C                     END                                End IF
C*
C                     READ WORKSTN                  3030 Read screen
C   KG                MOVE 'Y'       END     1           If cancel, set eoj
C*
C                     ENDSR
C*
C*-----------------------------------
C* Get and reformat library information
C*
C           GETLB     BEGSR
C*
C* Reformat fields
C*
C           LBENXT    IFEQ 'Y'                            If library extent
C                     MOVE 'YES'     LIBEXT  3              Yup
C                     ELSE                                Else
C                     MOVE ' NO'     LIBEXT                 Nope
C                     END                                 End IF
C*
C* Compute the number of members for each member type in the library,
C* and compute how many diskettes it would take to save this library
C*
C                     Z-ADD*ZEROS    MCNT                Clear member counts
C                     MOVE *BLANKS   MEMBER  8           Select next member
C                     Z-ADD1         M      10           Process objects 1st
C*
C           M         DOWLE4                             Count each mbr type
C                     MOVE MTYP,M    TYPE    1             Get member type
C                     MOVE ' '       RCODE                 Reset return code
C*
C           RCODE     DOUEQ'2'                             Stop at last mbr
C                     EXIT SUBRLD                            Get mbr info
C                     RLABL          INPLIB                  Library name
C                     RLABL          MEMBER                  Next member
C                     RLABL          TYPE                    Member type
C                     RLABL          MEMDS                   Output
C                     RLABL          RCODE   1               Return code
C           RCODE     IFNE '2'                               If not end
C                     ADD  1         MCNT,M                    incr. counter
C                     END                                    End IF
C                     END                                  End DO
C*
C                     ADD  1         M                     Another type
C                     END                                End DO
C*
C* Estimate approx  number of diskettes if library saved with SAVELIBR
C*
C           LBADIR    ADD  LBUDIR    LDTDIR  50          Total dir entries
C           LBUDIR    DIV  LDTDIR    USDPCT  33H         % director used
C           LBDRSZ    MULT USDPCT    TEMP84  84          Use directory
C                     Z-ADDTEMP84    LDUSEC  50            sectors
C                     SUB  LDUSEC    TEMP84     11       If partial sector
C   11                ADD  1         LDUSEC              assign it full sct
C*
C           LDUSEC    ADD  LBUSEC    LBTSEC  70          Total used sectors
C*
C           LBTSEC    MULT 2         TEMP7   70          2 I1   1 F1 sectors
C           TEMP7     DIV  1924      DK1S1D  30          Sectors/1S1D dskt
```

```
C                    MVR            TEMP84     11     If partial diskette
C   11               ADD  1         DK1S1D              round up
C*
C           LBTSEC   DIV  2         TEMP7             2 F1   1 I1 sectors
C           TEMP7    DIV  592       DK2S1D  30        Sectors/2S1D dskt
C                    MVR            TEMP84     11     If partial diskette
C   11               ADD  1         DK2S1D              round up
C*
C           LBTSEC   DIV  3848      DK1S2D  30        Sectors/1S2D dskt
C                    MVR            TEMP84     11     If partial diskette
C   11               ADD  1         DK1S2D              round up
C*
C           LBTSEC   DIV  4         TEMP7             2 F1 · 1 I1 sectors
C           TEMP7    DIV  1184      DK2S2D  30        Sectors/2S2D dskt
C                    MVR            TEMP84     11     If partial diskette
C   11               ADD  1         DK2S2D              round up
C*
C                    ENDSR
C*
C*-----------------------------------
C* Get and reformat member information
C*
C           GETMR    BEGSR
C*
C* Get sub-type description
C*
C                    Z-ADD1         TP      20        1st subtype desc
C           DRATR5   LOKUPSTA,TP                   11 Find subtype desc
C  N11               Z-ADD34        TP                Invalid subtype
C                    MOVE STD,TP    STDESC 28         Desc  to screen
C*
C* Get descriptions for each attribute byte and put in screen array.
C*
C                    MOVE *BLANKS   AL                Blank out list.
C                    MOVE DRATR1    AB,1              Set up attribute
C                    MOVE DRATR2    AB,2                byte array to
C                    MOVE DRATR3    AB,3                simplify
C                    MOVE DRATR4    AB,4                bit testing.
C                    MOVE DRATR6    AB,5
C                    Z-ADD1         ALP     20        1st screen line
C                    Z-ADD1         AP      10        1st attr byte
C*
C           AP       DOWLE5                           Look at each byte
C                    Z-ADD1         BP      10          1st bit in attr
C*
C           BP       DOWLE8                             Test each bit
C                    SETOF                       12
C                    TESTBB,BP      AB,AP        12       Is it on?
C*
C   12      1        IFEQ 1                                 Yes-Display it!
C           AP       SUB  1         ATP     20                Calc. index
C                    MULT 8         ATP                         into attr
C                    ADD  BP        ATP                         desc  array
C           ALP      IFLE 11                                  If on screen
C                    MOVE AT,ATP    AL,ALP                      desc->scrn
C                    ADD  1         ALP                         bump line
C                    END                                      End IF
C                    END                                    End IF
C*
C                    ADD  1         BP                      Next bit
C                    END                                  End DO
C*
C                    ADD  1         AP                  Next attr byte
C                    END                              End DO
C*
C* For SSP releases 1 thru 5, DRREL contains 01 thru 05
C* For SSP release 5.1, DRREL contains 51
C* Therefore, DRREL must be reformatted so that the releases go
C* from 1.0 thru 5.1
C*
C           DRREL    IFGT 9                           If rel  5.1
C           DRREL    DIV  10        DRRELA  21          move dec  left
C                    ELSE                             Else
C                    Z-ADDDRREL     DRRELA              add give it dec
C                    END                              End IF
C*
```

```
C* Adjust date field
C*
C                     MULT 100.0001  DRDATE           YYMMDD->MMDDYY
C*
C* Compute program size for object or subroutine members
C*
C           INPTYP    COMP 'O'                    11 If object or
C  N11      INPTYP    COMP 'R'                    11   If subroutine,
C   11      DRCORE    MULT 256       TEMP6   60            compute kbytes
C   11      TEMP6     DIV  1024      DRCORE    H           used
C*
C* Display MRT status for procedure
C*
C           INPTYP    IFEQ 'P'                       If procedure
C                     SETOF                       43   display MRT msg
C           DRMRTC    IFEQ 'FF'                        If MRT
C                     MOVE 'Yes'     MRT     3           then flag
C                     ELSE                             Else
C                     MOVE 'No'      MRT                 don't flag
C                     END                              End IF
C                     END                            End IF
C*
C           INPTYP    COMP 'S'                    43 If source, not MRT
C*
C                     ENDSR
C*
C*-----------------------------------
C* If user just pressed Enter and didn't enter new information, get the
C* next LIBRI request from POP's command list
C*
C           GETNX     BEGSR
C*
C* Check for new user input
C*
C           INPLIB    COMP SAVLIB                 11 Library change?
C   11      INPMBR    COMP SAVMBR                 11 Member change?
C   11      INPTYP    COMP SAVTYP                 11 Type change?
C  N11                MOVE 'Y'       NEWINF  1       New info entered
C   11                MOVE 'N'       NEWINF          or not entered
C*
C* Get either new user input or next POP command, as appropriate
C*
C           NEWINF    IFEQ 'Y'                       If new info
C                     EXSR SAVIN                       save it
C                     ELSE                           Else
C                     EXSR GETPC                       get POP command
C                     END                            End IF
C*
C* If not eoj, perform error check
C*
C           END       IFNE 'Y'                       If not eoj
C           REQST     CASEQ'LIBRONLY'CHKLB             Library case
C                     CAS            CHKMR             Member case
C                     END                              End CASE
C                     END                            End IF
C*
C* If still not eoj, retrieve lib or member info and display it
C*
C           END       IFNE 'Y'                       If not eoj
C           REQST     CASEQ'LIBRONLY'GETLB               Do lib case
C                     CAS            GETMR               or mem case
C                     END                              End CASE
C                     EXSR DSPSC                       Display screen
C                     END                            End IF
C*
C                     ENDSR
C*
C*-----------------------------------
C* Get the next "I" opcode from POP's command list
C* End program when no more commands in list
C*
C           GETPC     BEGSR
C*
C                     MOVE *BLANKS   PELEM           Clear element
C           LP        DOWLE12                        Do while els remain
C                     MOVE PCL,LP    PELEM             Get list element
```

```
C           OPCOD     IFEQ 'I'                          If 'I' opcode
C                     GOTO GETPC1                         Then exit loop
C                     END                               End IF
C                     ADD  1         LP                 Bump to next elt
C                     END                             End DO
C           GETPC1    TAG                             (loop exit point)
C*
C           LP        IFLE 12                         If got one
C                     EXSR GETPE                        Get POP element
C                     ADD  1         LP                 Bump to next elt
C                     ELSE                            Else
C                     MOVE 'Y'       END                Set eoj
C                     END                             End IF
C*
C           GETPCX    ENDSR
C*
C*-----------------------------------
C* Get the values from a single POP command list element
C*
C           GETPE     BEGSR
C*
C* If called from library (not member) screen, only library information
C* will be displayed in all LIBRI operations
C*
C           POPTYP    IFEQ 'L'                        If library request
C                     MOVE 'LIBRONLY'REQST   8          flag it
C                     MOVE '*LIBR   'INPMBR             then init for
C                     MOVE 'L'       INPTYP             SUBRLD fetch
C                     END                             End IF
C*
C* Set input fields accordingly
C*
C           REQST     IFEQ 'LIBRONLY'                 If library request
C                     MOVE POPLIB    INPLIB             make it new input
C                     ELSE                            Else member request
C                     MOVE MBRLIB    INPLIB  8          make it new input
C                     MOVE POPMBR    INPMBR  8
C                     MOVE POPTYP    INPTYP  1
C                     END                             End IF
C                     EXSR SAVIN                      Save input
C*
C                     ENDSR
C*
C*-----------------------------------
C* Save the input values for comparison purposes
C*
C           SAVIN     BEGSR
C*
C                     MOVE INPLIB    SAVLIB  8          Save library name
C                     MOVE INPMBR    SAVMBR  8          member
C                     MOVE INPTYP    SAVTYP  1          and object type
C*
C                     ENDSR
C*-----------------------------------
C/EJECT
OWORKSTN E                LIB
O                                    K8 'LIBRARY '
O                         INPLIB     8
O                         LBLBSZZ   13
O                         LBBLIB    19
O                         LBELIB    25
O                         LIBEXT    28
O                         LBFMT1    34
O                         LBDRSZZ   39
O                         LBBDIR    45
O                         LBUDIRZ   50
O                         LBEDIR    56
O                         LBADIRZ   61
O                         LBUSECZ   67
O                         LBBMEM    73
O                         LBASECZ   79
O                         LBEMEM    85
O                         MCNT,1Z   89
O                         LBNMEM    95
O                         MCNT,2Z   99
O                         MCNT,3Z  103
```

```
0526 O                         MCNT,4Z  107
0527 O                         DK1S1DZ  110
0528 O                         DK2S1DZ  113
0529 O                         DK1S2DZ  116
0530 O                         DK2S2DZ  119
0531 O*
0532 OWORKSTN E                MEMOR
0533 O                                   K8 'MEMBEROR'
0534 O                         INPLIB     8
0535 O                         INPMBR    16
0536 O                         INPTYP    17
0537 O                         DRATR5    19
0538 O                         DRMOD Z   25
0539 O                         STDESC    53
0540 O                         DRDATE    61 '  /  /  '
0541 O                         DRTOTLZ   64
0542 O                         DRTIME    69 '     '
0543 O                         DR#TXTZ   74
0544 O                         DRRELA    77 ' .0'
0545 O                         DRCOREZ   80
0546 O                         DRMRTNZ   82
0547 O                         DRADDR    88
0548 O                         DRLINK    92
0549 O                         DRSCA     96
0550 O                         DRRLD     98
0551 O                         DRPTF@   102
0552 O                         AL       520
0553 OWORKSTN E                MEMSP
0554 O                                   K8 'MEMBERSP'
0555 O                         INPLIB     8
0556 O                         INPMBR    16
0557 O                         INPTYP    17
0558 O                         DRATR5    19
0559 O                         DRMOD Z   25
0560 O                         STDESC    53
0561 O                         DRDATE    61 '  /  /  '
0562 O                         DRTOTLZ   64
0563 O                         DRTIME    69 '  .  '
0564 O                         DR#STMZ   74
0565 O                         DRRELA    77 '  0'
0566 O                         DR#TXTZ   80
0567 O                         MRT       83
0568 O                         DRADDR    89
0569 O                         AL       507
0570 OWORKSTN E                LIBERR
0571 O                                   K8 'LIBRERR '
0572 O                         INPLIB     8
0573 OWORKSTN E                MEMERR
0574 O                                   K8 'MEMBERR '
0575 O                         INPLIB     8
0576 O                         INPMBR    16
0577 O                         INPTYP    17
** Member types
ORPS
** Member sub-type descriptions    STA,STD element
02                         Data         1
11                Auto response         2
12                  Auto report         3
13             Basic procedures         4
14                          DFU         5
15                Screen format         6
16                         Menu         7
17               Message member         8
18                   Phone list         9
19                         Sort        10
31                    Assembler        11
32                        BASIC        12
33                        COBOL        13
34                      FORTRAN        14
35                          RPG        15
36                          WSU        16
37        Software distribution        17
40                  Unspecified        18
41      Business Graphics chart        19
42       Business Graphics data        20
43     Business Graphics format        21
```

```
51        SSP-ICF configuration       22
52         System configuration       23
53                Editable text       24
54               Free form text       25
55               Hard copy text       26
56X.25 pkt. switching link ctl        27
57Communications & system mgmt        28
58                        Query       29
59         Cross system product       30
5A             Query data entry       31
5B   Document library services        32
5C              Keys procedures       33
FF  Invalid sub-type specified        34
** Member attr. descriptions      AD element  Attr  byte  Attr. bit
SSP Attribute bit                     1           1          0
O-Privileged module / P-Don't log OCL 2           1          1
Non-inquirable module                 3           1          2
O-SFGR format load / P-Proc. with data 4          1          3
Source required                       5           1          4
Non base SSP module                   6           1          5
PTF has been applied                  7           1          6
Module has overlays                   8           1          7
Dedicated module                      9           2          0
Never ending program                 10           2          1
Module has OXRF fmt. index table     11           2          2
Security authorization required      12           2          3
Cannot load program with // LOAD     13           2          4
Program has common                   14           2          5
Prog  with utility control stmts     15           2          6
Module has OXRF WTG table            16           2          7
$WORK2 file required                 17           3          0
Do not swap this task                18           3          1
High level of dedication             19           3          2
Program needs FORTRAN micro-code     20           3          3
Member is a configuration record     21           3          4
Member must be transferred to        22           3          5
Member cross-referencable            23           3          6
New copy of MRT program required     24           3          7
Program needs BASIC micro-code       25           4          0
Pad module (spaceholder)             26           4          1
SUNGLOW program                      27           4          2
IBM supplied program                 28           4          3
Resides in library extent            29           4          4
DDS load format member               30           4          5
System transient member              31           4          6
One copy execution only              32           4          7
Dynamically privileged               33           6          0
Does not need swap area              34           6          1
Emulation member                     35           6          2
Has memory resident overlays         36           6          3
PC LAN microcode member              37           6          4
Not a valid attribute bit            38           6          5
Not a valid attribute bit            39           6          6
Not a valid attribute bit            40           6          7
/*
```

Figure 14-2

Screen format member LIBRIFM

```
*.      1 . .   . 2      .. 3   . ... 4  .  ... 5 ... ..  6 ... ... 7 ... ... 8
   S**********************************************************************
   S* SCREEN NAME...... LIBRIFM
   S* DESCRIPTION   . . LIBRI
   S* PROGRAMMER... .   Chuck Lundgren (Iris Software, Inc.)
   S* DATE WRITTEN. ... June 1989
   S*
   S* VERSION DATE      FIX DESCRIPTION
   S* ---- -----  ------------------------
   S**********************************************************************
   SLIBRARY          97 YY        Y                           G
   S**********************************************************************
   S* FORMAT NAME..     LIBRARY
   S* PURPOSE....... .  Library information screen
   S**********************************************************************
   D           4 1 2Y                                     CLibr
   DLNAME      8 1 7Y  Y             Y          Y
```

```
D           1 178Y  Y          Y     Y           CO
D           7 3 2Y                               CLIBRARY
D          26 345Y                               CLIBRARY ADDRESSES (in hX
Dex)
D           4 4 4Y                               CSize
DLBLBSZ     5 428Y               Y
D           6 434Y                               CBlocks
D          12 447Y                               CFirst sector
DLBBLIB     6 466Y               Y
D          11 547Y                               CLast sector
DLBELIB     6 566Y               Y
DLIBEX     13 5 4Y                               CExtent active
DEXTAVL     3 530Y               Y
DVTITLE    10 647Y                                VTOC entry
DLBFMT1     6 666Y               Y
D          17 8 2Y                               CLIBRARY DIRECTORY
D          36 845Y                               CLIBRARY DIRECTORY ADDREX
DSSES (in hex)
D           4 9 4Y                               CSize
DLBDRSZ     5 928Y               Y
D           7 934Y                               CSectors
D          12 947Y                               CFirst sector
DLBBDIR     6 966Y               Y
D          1210 4Y                               CUsed entries
DLBUDIR     51028Y               Y
D          111047Y                               CLast sector
DLBEDIR     61066Y               Y
D          1711 4Y                               CAvailable entries
DLBADIR     51128Y               Y
D          1513 2Y                               CLIBRARY MEMBERS
D          331345Y                               CLIBRARY MEMBER ADDRESSEX
DS (in hex)
D           414 4Y                               CUsed
DLBUSEC     61427Y               Y
D           71434Y                               CSectors
D          121447Y                               CFirst sector
DLBBMEM     61466Y               Y
D           915 4Y                               CAvailable
DLBASEC     61527Y               Y
D           71534Y                               CSectors
D          111547Y                               CLast sector
DLBEMEM     61566Y               Y
D          1416 4Y                               CObject members
DDIROB      41629Y               Y
D          181647Y                               CNext avail  sector
DLBNMEM     61666Y               Y
D          1817 4Y                               CSubroutine members
DDIRPC      41729Y               Y
D          1718 4Y                               CProcedure members
DDIRSB      41829Y               Y
D          341845Y                               CNO  DISKETTES REQ  TO SX
DAVE LIBRARY
D          1419 4Y                               CSource members
DDIRSC      41929Y               Y
D          201947Y                               C1S1D (128K) diskette
DDK1S1D     31969Y               Y
D          202047Y                               C2S1D (256K) diskette
DDK2S1D     32069Y               Y
D          202147Y                               C1S2D (512K) diskette
DDK1S2D     32169Y               Y
D          202247Y                               C2S2D (1.2M) diskette
DDK2S2D     32269Y               Y
DDK2S2D     6524 2Y              Y               CEnter new library name X
Dor press Cmd-7 to return to library screen
SMEMBEROR             YY       Y                            G
S*****************************************************************
S* FORMAT NAME       MEMBEROR
S* PURPOSE           Object/Subroutine member information screen
S*****************************************************************
D           4 1 2Y                               CLibr
DLIBNAM     8 1 7Y  Y            Y         Y
D           6 116Y                               CMember
DMBRNAM     8 123Y  Y            Y         Y
D           4 132Y                               CType
DMBRTYP     1 137Y  YA           Y         Y
D           1 178Y  Y          Y     Y           C1
```

```
D           6 3 2Y                                     CMEMBER
D          14 340Y                                     CVERSION STATUS
D           8 4 4Y                                     CSub-type
DMBRSUB     2 430Y              Y
D          16 442Y                                     CReference number
DDRREL      6 461Y              Y
DSTDESC    28 5 4Y              Y
D          12 542Y                                     CDate changed
DDRDATE     8 559Y              Y
D           4 6 4Y                                     CSize
DDRTOTL     3 629Y              Y
D           7 633Y                                     CSectors
D          12 642Y                                     CTime changed
DDRTIME     5 662Y              Y
D          12 7 4Y                                     CText sectors
DDR#STM     5 727Y              Y
D           7 733Y                                     CSectors
D          13 742Y                                     CRelease level
DDRREL      3 764Y              Y
D          12 8 4Y                                     CProgram size
DDRCORE     3 829Y              Y
D           7 833Y                                     CK bytes
D          14 9 4Y                                     CMRT max  count
DDRMRT      2 930Y              Y
D          2511 2Y                                     CMEMBER ADDRESSES (in heX
Dx)
D          101140Y                                     CATTRIBUTES
D          1912 4Y                                     CFirst member sector
DMBRADR     61226Y              Y
D           913 4Y                                     CLink edit
DDRLINK     41328Y              Y
D          1114 4Y                                     CEntry point
DDRSCA      41428Y              Y
D          1615 4Y                                     CRLD displacement
DDRRLD      21530Y              Y
D          2216 4Y                                     CPTF table displacement
DDRPTF@     41628Y              Y
DATR,1     381242Y              Y
DATR,2     381342Y              Y
DATR,3     381442Y              Y
DATR,4     381542Y              Y
DATR,5     381642Y              Y
DATR,6     381742Y              Y
DATR,7     381842Y              Y
DATR,8     381942Y              Y
DATR,9     382042Y              Y
DATR,10    382142Y              Y
DATR,11    382242Y              Y
D          7424 2Y              Y                      CEnter new library or meX
Dmber name, or member type, or press Cmd-7 to return
SMEMBERSP              YY       Y                              G
S*******************************************************************
S* FORMAT NAME         MEMBERSP
S* PURPOSE.            Source/Procedure member information screen
S*******************************************************************
D           4 1 2Y                                     CLibr
DLIBNAM     8 1 7Y  Y           Y         Y
D           6 116Y                                     CMember
DMBRNAM     8 123Y  Y           Y         Y
D           4 132Y                                     CType
DMBRTYP     1 137Y  YA          Y         Y
D           1 178Y  Y         Y     Y                  C1
D           6 3 2Y                                     CMEMBER
D          14 340Y                                     CVERSION STATUS
D           8 4 4Y                                     CSub-type
DMBRSUB     2 430Y              Y
D          16 442Y                                     CReference number
DDRREL      6 461Y              Y
DSTDESC    28 5 4Y              Y
D          12 542Y                                     CDate changed
DDRDATE     8 559Y              Y
D           4 6 4Y                                     CSize
DDRTOTL     3 629Y              Y
D           7 633Y                                     CSectors
D          12 642Y                                     CTime changed
DDRTIME     5 662Y              Y
```

```
D           10 7 4Y                                   CStatements
DDR#STM      5 727Y               Y
D           13 742Y                                   CRelease level
DDRREL       3 764Y               Y
D           13 8 4Y                                   CRecord length
DMBRLTH      3 829Y               Y
D           11 9 4Y                  43               CMRT Maximum
DMRT         3 929Y               Y  43
D           2511 2Y                                   CMEMBER ADDRESSES (in heX
Dx)
D           101140Y                                   CATTRIBUTES
D           1912 4Y                                   CFirst member sector
DMBRADR      61226Y               Y
DATR,1      381242Y               Y
DATR,2      381342Y               Y
DATR,3      381442Y               Y
DATR,4      381542Y               Y
DATR,5      381642Y               Y
DATR,6      381742Y               Y
DATR,7      381842Y               Y
DATR,8      381942Y               Y
DATR,9      382042Y               Y
DATR,10     382142Y               Y
DATR,11     382242Y               Y
D           7424 2Y               Y                   CEnter new library or meX
Dmber name, or member type, or press Cmd-7 to return
SLIBRERR              YY      Y                              G
S*****************************************************************
S* FORMAT NAME        LIBRERR
S* PURPOSE            Library error screen
S*****************************************************************
D           47 1 2Y                                   CUnable to find the follX
Dowing library       NAME
DLIBNAM      8 152Y  Y            Y         Y
D            1 178Y  Y         Y     Y                CO
D           70 4 2Y               Y                   CEnter new library name X
Dor press Command 7 to return to library screen
SMEMBERR              YY      Y                              G
S*****************************************************************
S* FORMAT NAME        MEMBERR
S* PURPOSE            Member error screen
S*****************************************************************
D           37 1 2Y                  40               CUnable to find the follX
Dowing library
DLIBNAM      8 152Y  Y            Y         Y
D           76 5 2Y               Y                   CEnter new library, membX
Der name or type, or Cmd-7 to return to library screen
D           36 2 2Y                  41               CUnable to find the follX
Dowing member
DSMNAME      8 252Y  Y            Y         Y
D            5 344Y                                   CType
DSOTYPE      1 352Y  Y            Y         Y
D            1 378Y  Y         Y     Y                C1
D            7 242Y                                   CMember
D            8 141Y                                   CLibrary
D           34 3 2Y                  42               CMember type must be O, X
DR, P, or S
```

Figure 14-3

Procedure LIBRI

```
*  Procedure    LIBRI
*  Parameters   For library information      For member information
*     1            Library name                  Member name
*     2            L                             Member type (O,R,P, or S)
*     3            O                             Library name
*
// LOAD LIBRI
// RUN
```

Note: Since #POPLIB already contains a LIBRI (save library) procedure, you should rename either it or the LIBRI (information) procedure before installing this procedure in #POPLIB.

Figure 14-4a

Library information screen

```
 Libr #LIBRARY

 LIBRARY                                 LIBRARY ADDRESSES (in hex)
   Size                     7171 Blocks    First sector       003ADF
   Extent active              NO           Last sector        0152FC]
                                           VTOC entry         001A0F

LIBRARY DIRECTORY                       LIBRARY DIRECTORY ADDRESSES (in
hex)
   Size                      768 Sectors   First sector       003AE0
   Used entries             1918           Last sector        003DDE
   Available entries        1916

LIBRARY MEMBERS                         LIBRARY MEMBER ADDRESSES (in
hex)
   Used                    66994 Sectors   First sector       003DDF
   Available                3408 Sectors   Last sector        0152C5
   Object members           1418           Next avail  sector 014576
   Subroutine members          9
   Procedure members         447         NO  DISKETTES REQ  TO SAVE
LIBRARY
   Source members             44           1S1D (128K) diskette   71
                                           2S1D (256K) diskette   57
                                           1S2D (512K) diskette   18
                                           2S2D (1 2M) diskette   15

 Enter new library name or press Cmd-7 to return to library screen
```

Figure 14-4b

Source and procedure member information screen

```
Libr POM      Member LIBRI     Type S

MEMBER                              VERSION STATUS
   Sub-type                 40         Reference number        4
                   Unspecified         Date changed      7/11/89
   Size                     80 Sectors Time changed        15:58
   Statements              650         Release level         5.1
   Record length            96

MEMBER ADDRESSES (in hex)           ATTRIBUTES
   First member sector   0269E6        SUNGLOW program

Enter new library or member name, or member type, or press Cmd-7 to return
```

Figure 14-4c

Object and subroutine member information screen

```
Libr POM      Member LIBRI    Type O

MEMBER                              VERSION STATUS
  Sub-type                   35       Reference number        4
                            RPG       Date changed      7/11/89
  Size                       80 Sectors  Time changed     15:59
  Text sectors               80 Sectors  Release level      5.1
  Program size               20 K bytes
  MRT max  count

MEMBER ADDRESSES (in hex)           ATTRIBUTES
  First member sector   026A36        Privileged module
  Link edit               0000        SUNGLOW program
  Entry point             0000
  RLD displacement          CA
  PTF table displacement  0000


Enter new library or member name, or member type, or press Cmd-7 to return
```

Figure 14-5a

Library error screen

```
Unable to find the following library      NAME    JUNKO

Enter new library name or press Command 7 to return to library screen
```

Figure 14-5b

Library member error screen

```
                                              Library     POM
Unable to find the following member            Member     LIBRZ
                                                 Type     0


Enter new library, member name or type, or Cmd-7 to return to library
screen
```

Figure 14-6

POP command list

51	52 - 59	60	← LDA positions
P	LIBTEST	L	Command 1
61	62 - 69	70	
J	#IDALIB	L	Command 2

Figure 14-7

Command list before compression

51	52 - 59	60	← LDA positions
P	LIBTEST	L	Command 1 ← Already executed
61	62 - 69	70	
I	#IDALIB	L	Command 2 ← Currently executing
71	72 - 79	80	
J	#VDSKLIB	L	Command 3 ← Pending execution
81	82 - 89	90	
I	FSLIB	L	Command 4
91	92 - 99	100	
K	DFULIB	L	Command 5

Figure 14-8

Compressed command list

51	52 - 59	60	← LDA positions
P	LIBTEST	L	Command 1 ← Already executed
61	62 - 69	70	
I	#IDALIB	L	Command 2 ← Already executed
71	72 - 79	80	
J	#VDSKLIB	L	Command 3 ← Pending execution
81	82 - 89	90	
K	DFULIB	L	Command 4

Editing in Two FSEDIT Sessions

by Mark Lazarus and Abraham Notik

Code on diskette:
Procedure code FSEDIT2S

A typical solution to the problem of editing multiple sessions with FSEDIT (POP's editor) is to replace the #POPLIB FSEDIT procedure statement in Figure 14-9a with the lines of code in Figure 14-9b. The problem with this approach is that it doesn't cover *all* possible situations. Consider the following scenario: You start up session 1 and press the Attention key and select option 1 to run an inquiry session. You fire up session 2 and then Attention key/option 7 to get back to session 1. Now, if you press the Attention key, select option 3, and cancel session 1, and then try to start FSEDIT again, it bombs. This is because the session 1 work file is still on disk, and FSEDIT attempts to recover the still-active session 2 work file.

A problem also occurs if you've exited session 1 and the terminal goes down or the edit session gets interrupted while session 2 is still active. When your system comes back up, FSEDIT creates the session 1 work file and ignores the fact that there is another session (session 2) to be recovered.

An improved solution modifies the FSEDIT procedure by replacing the original statement (Figure 14-9a) with the lines of code in Figure 14-9 c. This solution lets you toggle back and forth between the two editing sessions without running into problems.

Figure 14-9a

Original FSEDIT procedure statement

```
// IF DATAF1-?1OF'FS?L'214,4'??WS?'?  FSEDRCVR *ALL
```

Figure 14-9b

A typical modification to the FSEDIT procedure

```
// IF  OATAF1-FS?L'214,4'??WS?  EVALUATE P10-FI?L'214,4'??WS?
// IFF OATAF1-FS?L'214,4'??WS?  EVALUATE P10-FS?L'214,4'??WS?
// IF  DATAF1-?10?  FSEDRCVR *ALL
```

Figure 14-9c

An improved modification to the FSEDIT procedure

```
// EVALUATE P10-FI?L'214,4'??WS?
// IF  ?F'A,?10?'?-O  GOTO SKIP1
// FILE NAME-?10?,WAIT-NO
// IF ?CD?-0000          FSEDRCVR *ALL
?64?
•_
// TAG SKIP1
// EVALUATE P10-FS?L'214,4'??WS?
// IF  ?F'A,?10?'?-O  GOTO SKIP2
// FILE NAME-?10?,WAIT-NO
// IF ?CD?-0000          FSEDRCVR *ALL
// TAG SKIP2
?64?
•_
// FILE NAME-?1OF'FS?L'214,4'??WS?'?,WAIT-NO
// IFF ?CD?-0000  EVALUATE P10-FI?L'214,4'??WS?
```

Note: This code can be found in procedure FSEDIT2S on the diskette.

Emulating RPGONL and COBOLONL in POP

by Alvaro de Leon

Code on diskette:
Procedures LIBRL, LIBRO2

An on-line programming system can accelerate the development of your programs because in an on-line programming environment, you can enter a new program (or make a number of changes in an existing program) and then compile it, view the error messages on the screen, do the necessary editing, and immediately compile the corrected version. The result is that your programs go into production more readily.

The IBM-supplied procedures that support on-line development of COBOL and RPG programs are COBOLONL and RPGONL. These procedures use the DSM (Diagnose Source Member) parameter in the S/36 COBOLC and RPGC commands to record diagnostic messages in an easy-to-read format so you can use SEU to read and correct the errors.

Because I prefer the full-screen editor (FSEDIT) of IBM's POP to SEU, I wrote two procedures, LIBRO and LIBRL, that alternate FSEDIT and the given language compiler help screens (see Figures 14-10 and 14-11, respectively). With these procedures saved in #LIBRARY, you can enter O to iteratively edit/compile an RPG program or use L to iteratively edit/compile a COBOL program. When the program source code is satisfactory, you can exit either procedure by pressing Command key 7 or Command key 19 from the edit screen and then Command key 3 from the compiler help screen.

These procedures provide the functions of RPGONL and COBOLONL, with the added feature of full-screen editing, all in just four lines of code (compared with more than 260 lines in the RPGONL procedure and more than 420 lines in the COBOLONL procedure).

Figure 14-10
Procedure LIBRO

```
// TAG INICIO
FSEDIT     ?1?,R,?L'1,8'?
HELP RPGC  ?1?,?L'1,8'?,DSM,NOPRINT,NOXREF
// IFF ?CD?-2143  GOTO INICIO
```

Note: This procedure is named LIBRO2 on diskette. To use it in #POPLIB, you must rename it to LIBRO. Another procedure described in Transmitting Library Members via ODF/36 and POP, *page 415, uses the name LIBRO, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than O.*

Figure 14-11
Procedure LIBRL

```
// TAG INICIO
FSEDIT         ?1?,R,?L'1,8'?
HELP COBOLC    ?1?,?L'1,8'?,,DSM,NOPRINT,NOXREF
// IFF ?CD?-2143   GOTO INICIO
```

Removing Diagnostics from RPG Programs

by Manuel Humberto

Code on diskette:
Procedure LIBRM

FSEDIT can't remove diagnostic messages from source code when you compile programs using the DSM (Diagnose Source Member) parameter on the RPGC procedure. However, you can delete these diagnostic messages with utility LIBRM.

To install the utility using POP, place LIBRM (Figure 14-2) in #LIBRARY or #POPLIB. To execute the utility, put an M at the left of the names of the source members from which you want to remove the DSM diagnostic message lines (i.e., lines containing ?? in the first two positions). The LIBRM procedure then removes the diagnostic message lines from the selected source members.

Figure 14-12
Procedure LIBRM

```
// INFOMSG      NO
// IFF          SOURCE-'?1?,?3?'    MSG      ?WS?,No Source found ?1?
// IFF          SOURCE-'?1?,?3?'    RETURN
// IF           ?2?-                EVALUATE P2-S
// IF           DATAF1-LIBRM?WS?    DELETE   LIBRM?WS?,F1
```

```
// IF          DATAF1-LIBR1?WS?    DELETE   LIBR1?WS?,F1
FROMLIBR       ?1?,S,LIBRM?WS?,F1,T,100,?3?,...,96
COPYDATA       LIBRM?WS?,,LIBR1?WS?,.....,OMIT,1,EQ,'??'
TOLIBR         LIBR1?WS?,F1,,REPLACE,?3?,...,?1?,SOURCE
DELETE         LIBRM?WS?,F1
DELETE         LIBR1?WS?,F1
```

Blanking Out Columns 1-5 and 75-80 in RPG Source with POP

by Hermann Revilla Gutierrez

Code on diskette:
Procedure LIBRQ
RPG program LIBRQ
Screen format member LIBRQFM

S/36 programmers usually serialize their source RPG II programs either by using options 3 or 4 of the last SEU prompt or by answering the Serialize Member option on the Source Replacement Options prompt on POP's editor affirmatively. Some programmers also duplicate the program's name in columns 75 through 80.

A serialized program is useful during development, but storing it permanently wastes considerable disk space. The S/36 utility LIBRQ — consisting of procedure LIBRQ (Figure 14-13), program LIBRQ (Figure 14-14), and a display screen LIBRQFM (Figures 14-15a and 14-15b) — quickly deletes the serialization of a source member.

To execute utility LIBRQ using POP, place the utility in #LIBRARY by specifying the source member in a given library. You can either key Q at the left of the name(s) of the program(s) from which you want to remove the serialization or place the utility in any user's library. The user can execute the utility by running procedure LIBRQ. Procedure LIBRQ prompts you for the name, type, and library of the source member from which you want to delete the serialization.

Figure 14-13
Procedure LIBRQ

```
// IF ?2?/ EVALUATE P2=S
// IF ?1?/ PROMPT MEMBER-LIBRQFM,FORMAT-A,LENGTH-'8,1,8',START-1
// IF ?CD?/2007 RETURN
// IFF ?2?/S RETURN
// IF DATAF1-LIBRQ?WS? DELETE LIBRQ?WS?,F1
FROMLIBR ?1R?,SOURCE,LIBRQ?WS?,F1,T,80,?3R?,...,120
// * ' Ejecuci"n programa      LIBRQ'
// LOAD LIBRQ
// FILE NAME-LIBRQ,LABEL-LIBRQ?WS?
// RUN
TOLIBR LIBRQ?WS?,F1,,REPLACE,?3?,...,?1?,SOURCE
DELETE LIBRQ?WS?,F1
```

Note: Another procedure described in Putting a Job on the Job Queue from POP, *page 430, uses the name LIBRQ, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.*

Figure 14-14

Program LIBRQ

```
*... ... 1 ... .. 2 ... ... 3 ... ... 4 .. ... 5 .. .. 6 . . ... 7 ... ... 8
H     002                           B       1                             LIBRQ
F*************************************************************************
F*  Program:  LIBRQ     Written by: Ing. Hermann Revilla Gtz.            *
F*                                                                       *
F*  This program ends the serialization of a source member               *
F*  previously serialized using either SEU or POP's editor.              *
F*                                                                       *
F*************************************************************************
FLIBRQ   UP  F3840 120            DISK
ILIBRQ   NS  01    1NC/
I       AND        1NC?
I                                        1 120 REGIST
I                                        1   3 ASTER
I                                        1   5 SERIE
I                                        6   6 CAR
I                                       75  80 NOMPRO
I        NS  02    1 C/
I       OR         1 C?
I                                        1 120 REGIST
C   01 30              EXSR UNO
C   01N30              EXSR DOS
*****************************************************
C           UNO       BEGSR
C           NOMPRO    COMP NOMAUX                    40
C   40                MOVE *BLANK    NOMPRO
C                     MOVE *BLANK    SERIE
C           ASTER     COMP '** '                     LR
C   LR                SETOF                      0102
C                     ENDSR
*****************************************************
*****************************************************
C           DOS       BEGSR
C           CAR       COMP 'H'                       30
C   30                MOVE NOMPRO    NOMAUX  6
C   30                MOVE *BLANK    SERIE
C                     ENDSR
*****************************************************
OLIBRQ   D         01
O       OR         02
O                          REGIST   120
O                  01      SERIE      5
O                  01      NOMPRO    80
```

Figure 14-15a

Prompt screen LIBRQFM

```
DELETE SERIALIZATION PROGRAM

             Member name . . . . . . . .   . . . .         .

             Member type . . . . . . . . . .   . . .         . S

             Library containing member . . .          . .  .




                                Cmd 7 - CANCEL
```

Figure 14-15b
Screen format member LIBRQFM

```
* .. .. 1  .  ... 2 ...  .. 3  .  ... 4 . .  .. 5 .. .. 6 ..    7 ..  ... 8
  SA         0124      Y                                G
  DFL0001    31 326Y                Y                 CQUITER SERIALIZATION PRX
  DOGRAM
  DFL0002    49 714Y                                  CMember Name .. .   ....X
  D ...      . .. . .  .
  DFL0003     8 764Y  Y             Y        Y
  DFL0004    49 914Y                                  CMember Type   .    . . .X
  D..       ...  . . ..    . .
  DFL0005     1 964Y  Y             Y        Y
  DFL0006    491114Y                                  CLibrary Name that contaX
  Dins the member .      ...
  DFL0007     81164Y  Y             Y        Y
  DFL0008    232330Y                                  C<Mdato 7> CANCEL
```

Positioning LIBR to a Given Member

by Garry A. Abbott

Code on diskette:
Procedure POS

POP is a great facility for browsing and editing library members but leaves a bit to be desired in positioning the display of particular members. POP's requirement that you use the ? search command is especially a problem when you are using remote 5250 emulation on a PC.

To solve this problem, the procedure in Figure 14-16 substitutes three parameters into the LDA to cause POP to automatically bring up the member requested by parameter 2. Note that parameter 3 (member type) defaults to S.

Figure 14-16
Procedure POS

```
** QUICK LIBR DISPLAY SET ON PARMS
** P1- LIBRARY
** P2- MEMBER NAME
** P3- TYPE, S.P.O.R
// LOCAL OFFSET-1,DATA-'?1'?CLIB?'?    '
// LOCAL OFFSET-27,DATA-'YYYY         .I?3'S'???2?     '
// LIBRARY NAME-#POPLIB
// LIBR#
```

Transmitting Library Members via ODF/36 and POP

by Mike Otey

Code on diskette:
Procedures LIBRO, SENDODF
RPG programs ODFPOP, ODFGET
RPG code ODFSND, ODFMSG
Screen format member ODFPOPFM

ODF/36 is an IBM PRPQ that lets your S/36 transmit library members, files, job streams, and print spool files to remote systems using an APPC/APPN (Advanced Program-to-Program Communications/Advanced Peer-to-Peer Networking) communications link. You can operate ODF/36 either in interactive mode, by filling out prompt screens, or in batch mode, by calling procedures with parameter lists. The batch capabilities of ODF let you use ODF/36 in a primarily unattended environment and in your own utility procedures.

ODF/36's most useful feature is its ability to distribute application library maintenance from a central site to remote CPU locations by using the SENDLIBR procedure. The format of the ODF/36 SENDLIBR procedure is similar to the standard SSP LIBRLIBR procedure. Figure 14-17 illustrates the SENDLIBR procedure's interactive prompt screen. SENDLIBR is geared to work with individual library members, not arbitrary groups of members, and thus can be somewhat cumbersome to use. The first ODF/36 network management tool you need to construct is one that automates SENDLIBR's operation.

Figure 14-17

Send Library Members Through Network screen

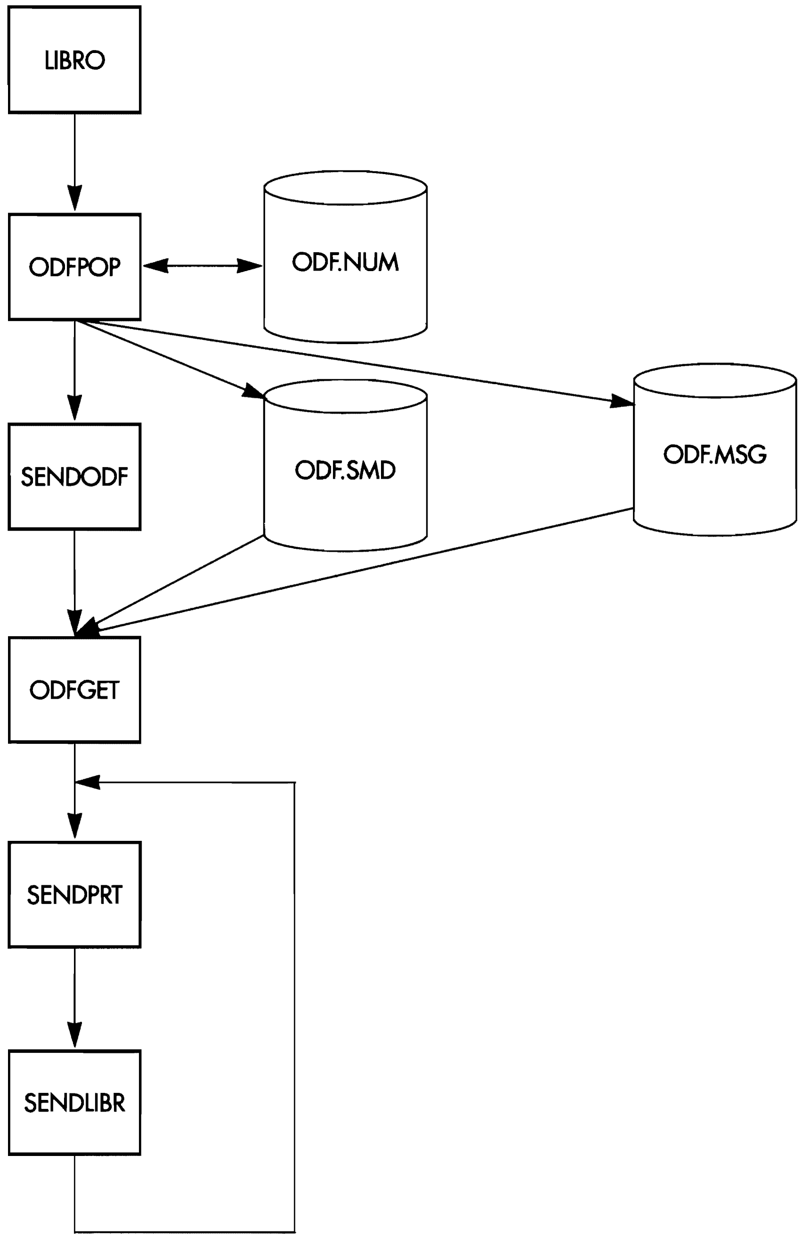
```
                SEND LIBRARY MEMBERS THROUGH NETWORK
Type choices, press Enter
ITEM                             CHOICE     POSSIBLE CHOICES
Member name                      ODFPOP     Name, partial name, ALL
 If partial name, enter ALL
Member type                      SOURCE     SOURCE, LOAD, PROC, LIBRARY,
                                            SUBR, PTF
Library                          #POPLIB
Format                           S36FMT     S36FMT, DATA, PUNCH
User                             OPERATOR
Address                          PORTLAND
Priority                         50         1-99
RSCS distribution code
RSCS class                       A          A-Z


Acknowledgment                   NOACK      NOACK, ACK






Cmd3-Go back            Cmd5-Add user list         Cmd7-End
```

An existing productivity tool that can help you improve the SENDLIBR user interface is IBM's POP utility. POP makes keying LIBRLIBR procedures obsolete, and it can do the same for ODF/36's SENDLIBR procedure. The "point and shoot" interface POP uses eliminates keying errors and lets you perform operations on arbitrary groups of objects as opposed to LIBRLIBR's — and SENDLIBR's — single-object orientation. By combining POP's library members interface and the batch mode of ODF/36, and by using the built-in user extendability, you can make two of IBM's most useful S/36 products complement one another.

Creating a New POP Opcode

The ODF/36 SENDLIBR procedure provides the engine to distribute the library members through the network, while POP provides a familiar and efficient user interface. To distribute library members to your remote sys-

Figure 14-18
ODFPOP overview



tems, you must add to the POP library members screen a new opcode that creates a connection between POP and ODF/36.

Implementing a new POP opcode doesn't require modifying any of the IBM-supplied POP procedures. Thus, it is relatively easy to implement the POP-ODF/36 send library members utility.

The first step in designing this send library members utility is to choose the new POP opcode. I chose O because it is an appropriate abbreviation for ODF, it is easy to remember, and I didn't already have an O function. Next, you must determine how this new opcode should work. To be consistent with the other POP opcodes, the utility should be able to perform operations on multiple members and should have the ability to display the selected members for confirmation first. Also, to make the most efficient use of the network, you should have the option to specify the distribution start time in case you want to schedule maintenance during low network-traffic periods. For audit purposes, you should be able to generate both a printed record of the maintenance at the central and the remote locations and an on-line record of each transmission to a remote location. Last, to alert the operators at the remote sites that they have received a maintenance update, you should be able to send an optional operator message with the transmission. You can also use this message to provide any additional instructions for the remote system operators.

The new O opcode consists of procedure LIBRO, workstation program ODFPOP and screen format member ODFPOPFM to display the library members selected for confirmation, file ODFSND to contain the selected members and message, program ODFGET to read the selected members out of the file for transmission, and, finally, procedure SENDODF to call the ODF/36 SENDLIBR and SENDPRT procedures to distribute the library members and their accompanying maintenance log and message. Figure 14-18 provides an overview of the system.

POP LIBR and LIBR# Procedures

Before you can understand the POP utility in detail, you need to understand the POP LIBR and LIBR# procedures (Figures 14-19a and 14-19b, respectively), which drive the POP library members display. The LIBR procedure displays library member lists and calls the LIBR# procedure from the POP LIBR procedure every time you enter an opcode other than B, N, or Y on the POP library members screen. The POP LIBR program handles opcodes B, D, N, and Y internally, but all other opcodes are passed into the LDA along with the associated member name and type (e.g., source, object). The LIBR# procedure is subsequently invoked and executes a command-handling procedure for each of the opcodes stored in the LDA (Figure 14-19b).

Procedure LIBR# follows a simple rule for determining the name of the procedure to execute for a given command: the name follows the form LIBR*x*, where LIBR is constant and *x* is the POP opcode. For example, procedure LIBR# processes an O opcode by executing a procedure named LIBRO. After procedure LIBR# processes the entire list of opcodes, it returns to the LIBR procedure to redisplay the library members list.

Figure 14-19a
POP's LIBR procedure

```
// LOCAL OFFSET-1,DATA-'?1'O'?
// LOCAL OFFSET-27,DATA-'YYYY           I
// LOCAL OFFSET-51,BLANK-120               CLEAR LDA BEFORE EXECUTING LIBR#
// RESET LIBR#
```

Figure 14-19b
POP's LIBR# procedure

```
// MEMBER USER1-LIBR##
// LOAD LIBR
* A SWITCH TO CLEAR THE SWITCH INDICATORS AFTER EDITING
// SWITCH 00000000
// RUN
LIBR?L'51,1'? ?L'52,8'?,?L'60,1'?,?L'1,8'?
LIBR?L'61,1'? ?L'62,8'?,?L'70,1'?,?L'1,8'?
LIBR?L'71,1'? ?L'72,8'?,?L'80,1'?,?L'1,8'?
LIBR?L'81,1'? ?L'82,8'?,?L'90,1'?,?L'1,8'?
LIBR?L'91,1'? ?L'92,8'?,?L'100,1'?,?L'1,8'?
LIBR?L'101,1'? ?L'102,8'?,?L'110,1'?,?L'1,8'?
LIBR?L'111,1'? ?L'112,8'?,?L'120,1'?,?L'1,8'?
LIBR?L'121,1'? ?L'122,8'?,?L'130,1'?,?L'1,8'?
LIBR?L'131,1'? ?L'132,8'?,?L'140,1'?,?L'1,8'?
LIBR?L'141,1'? ?L'142,8'?,?L'150,1'?,?L'1,8'?
LIBR?L'151,1'? ?L'152,8'?,?L'160,1'?,?L'1,8'?
LIBR?L'161,1'? ?L'162,8'?,?L'170,1'?,?L'1,8'?
// DEALLOC UNIT-I1
// RESET LIBR#
```

Figure 14-20
Procedure LIBRO

```
// LOAD ODFPOP
// FILE NAME-ODFSND,LABEL-ODF.SND,DISP-SHRMM
// FILE NAME-ODFMSG,LABEL-ODF.MSG,DISP-SHRMM
// FILE NAME-ODFSTAT,LABEL-ODF.NUM,DISP-SHRMM
// RUN
// IFF SWITCH8-1 EVOKE SENDODF
```

Note: Another procedure described in Emulating RPGONL and COBOLONL in POP, *page 411, uses the name LIBRO, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than O. If you do rename LIBRO, you must also change line 133 in the ODFPOP program to reflect the change. For example, if you rename LIBRO to LIBRU, you must change line 133 to* "`C      IFEQ      'U'`".

For my new O opcode, I created procedure LIBRO (Figure 14-20) in #POPLIB. Each time I enter an O on the POP library members screen, the O opcode is joined with the literal "LIBR" to form a new procedure — the LIBRO procedure. Procedure LIBRO then simply calls the workstation program ODFPOP to display the selected members and conditionally evokes the SENDODF procedure, which sends the library members.

Now two problems become evident: first, if we stick to our original design of sending up to 12 members at a time, we need a way to prevent the LIBRO procedure from being called multiple times; and second, we need a way to handle mixed opcodes on one screen. By using a combination of logic within the ODFPOP program and an additional procedure (LIBR*), we can solve these problems.

Figure 14-21a

Program ODFPOP

```
*    .  1    .   2 ..  .. 3  .  .. 4 .      5      . 6  .      7       8
H      64                        B                                  ODFPOP
F*******************************************************************
F*------------PROGRAM DESCRIPTION-------------
F*
F* THIS PROGRAM UPDATES THE FIELDS OF A MACHINE RECORD
F*
F*******************************************************************
F*------------MAINTENANCE SUMMARY-------------
F*
F*  04/13/89 - MJO - PROGRAM WRITTEN
F*  08/31/89 - MJO - MODIFIED FRO NEWS 3X/400
F*******************************************************************
F*-------------FILE DESCRIPTION-------------
F*
FWORKSTN CD  F     618            WORKSTN
FODFSTAT UF  F 256 256            DISK
FODFSND  OF  F  32  32  4AI     1 DISK                    A
FODFMSG  OF  F 512 512  4AI     1 DISK                    A
F*
F*******************************************************************
F*------------INDICATOR SUMMARY--------------
F*
F********** INDICATORS ***********
F*
F*     01  -  INPUT SCREEN1
F*
F*     97  -  READ WORKSTATION END OF FILE
F*
F********** COMMAND KEYS ***********
F*
F*     KG  -  CMD 7  · END PROGRAM
F*
F*
I*******************************************************************
I*------------ARRAYS AND TABLES  -------------
E                    POP        12  1               POP CODE
E                    OBJ        12  8               OBJECT NAME
E                    TYP        12  1               OBJECT TYPE
E                    MSG        12 40               MSG ARRAY
E                    OT         12 10               CODE(1)-OBJ(10)-TYP(1)
I*******************************************************************
I*------------FILE SPECIFICATIONS-------------
I*
I***************************
I* WORKSTATION SCREENS
I***************************
I*
IWORKSTN NS  01
I* FORMAT-ODF
I                                        1   8 SYSTEM
I                                        9  16 USER
I                                       17  2200TIME
I                                       23  30 OBJ,1
I                                       31  31 TYP,1
I                                       32  71 MSG,1
I                                       72  79 OBJ,2
I                                       80  80 TYP,2
I                                       81 120 MSG,2
I                                      121 128 OBJ,3
I                                      129 129 TYP,3
I                                      130 169 MSG,3
I                                      170 177 OBJ,4
I                                      178 178 TYP,4
I                                      179 218 MSG,4
I                                      219 226 OBJ,5
I                                      227 227 TYP,5
I                                      228 267 MSG,5
I                                      268 275 OBJ,6
I                                      276 276 TYP,6
I                                      277 316 MSG,6
I                                      317 324 OBJ,7
I                                      325 325 TYP,7
I                                      326 365 MSG,7
I                                      366 373 OBJ,8
```

```
I                                      374 374 TYP,8
I                                      375 414 MSG,8
I                                      415 422 OBJ,9
I                                      423 423 TYP,9
I                                      424 463 MSG,9
I                                      464 471 OBJ,10
I                                      472 472 TYP,10
I                                      473 512 MSG,10
I                                      513 520 OBJ,11
I                                      521 521 TYP,11
I                                      522 561 MSG,11
I                                      562 569 OBJ,12
I                                      570 570 TYP,12
I                                      571 610 MSG,12
F*
IODFSTAT NS  98
I                                        1  400DF#
I            DS
I                                        5  12 OBJNAM
I                                       13  13 OBJTYP
I                                       14  190SDATE
I                                       20  250STIME
I*
I            DS
I                                        5  12 SYSTEM
I                                       13  20 OBJLIB
I                                       21 260 MSG1
I                                      261 500 MSG2
I                                      501 508 USER
IOTDS        DS
I                                        1  10 COT
I                                        1   1 C
I                                        2   9 O
I                                       10  10 T
F* POP LDA SPECIFICATIONS
IPOPDS      UDS
I                                        1   8 LIB
I                                       51 170 OT
I                                      301 3060OTIME
I                                      307 3100HOLD40
I*
C********************************************************************
C*----------CALCULATION SPECIFICATIONS-----------
C*
C           ONCE      DO   0         ONCE     10
C*
C                     TIME           STIME    60         SET UP INITIAL
C                     MOVE UDATE     SDATE    60          VALUES
C                     MOVE '1'       ON       1
C                     MOVE '0'       OFF      1
C                     Z-ADD000000    OTIME
C                     MOVEL'OPERATOR'USER
C                     SETOF                      U8
C*
C                     Z-ADD01        Y        20
C                     DO   12        X        20
C           OT,X      IFNE *BLANK
C                     MOVE OT,X      COT
C           C         IFEQ '0'
C                     MOVE C         POP,Y
C                     MOVE O         OBJ,Y
C                     MOVE T         TYP,Y
C                     ADD  01        Y
C                     MOVE *BLANK    OT,X
C                     MOVEL'*'       OT,X
C                     END
C*
C                     END
C                     END
C*
C                     EXCPTS1                                SHOW SCREEN FOR UP-
C*                                                            DATE
C                     END
C*
C***-------------------------***
```

```
C*
C                     READ WORKSTN                    97
C*
C   KG                DO
C                     SETON                       LRU8
C                     GOTO END
C                     END
C*
C           0000001   CHAINODFSTAT                50
C                     Z-ADDODF#      HOLD40  40
C           HOLD40    ADD  1         ODF#
C                     Z-ADDODF#      HOLD40
C                     EXCPTODFST
C*
C                     MOVE LIB       OBJLIB
C*
C                     DO   12        X       20
C           OBJ,X     IFNE *BLANK
C                     MOVE OBJ,X     OBJNAM
C                     MOVE TYP,X     OBJTYP
C                     EXCPTODFSRC
C                     END
C                     END
C*
C                     MOVEAMSG,1     MSG1
C                     MOVEAMSG,7     MSG2
C*
C                     EXCPTODFTXT
C                     SETON                       LR
C*
C*
C***-------------------------***
C*
C*          ***       ***
C           END       TAG
C*          ***       ***
C*
C***-------------------------***
C*
C*
O*********************************************************************
O*----------FILE OUTPUT SPECIFICATIONS----------
O*
O********************
O* WORKSTATION OUTPUT
O********************
O*
OWORKSTN E                S1
O                                   K8 'ODF
O                         LIB        8
O                         SYSTEM    16
O                         USER      24
O                         OTIME     30
O                         OBJ,1     38
O                         TYP,1     39
O                         MSG,1     79
O                         OBJ,2     87
O                         TYP,2     88
O                         MSG,2    128
O                         OBJ,3    136
O                         TYP,3    137
O                         MSG,3    177
O                         OBJ,4    185
O                         TYP,4    186
O                         MSG,4    226
O                         OBJ,5    234
O                         TYP,5    235
O                         MSG 5    275
O                         OBJ,6    283
O                         TYP,6    284
O                         MSG,6    324
O                         OBJ,7    332
O                         TYP,7    333
O                         MSG,7    373
O                         OBJ,8    381
```

```
0225 0                             TYP,8      382
0226 0                             MSG,8      422
0227 0                             OBJ,9      430
0228 0                             TYP,9      431
0229 0                             MSG,9      471
0230 0                             OBJ,10     479
0231 0                             TYP,10     480
0232 0                             MSG,10     520
0233 0                             OBJ,11     528
0234 0                             TYP,11     529
0235 0                             MSG,11     569
0236 0                             OBJ,12     577
0237 0                             TYP,12     578
0238 0                             MSG,12     618
0239 0********************
0240 0* ODFSTAT FILE
0241 0********************
0242 0*
0243 OODFSTAT E                    ODFST
0244 0                             ODF#         4
0245 0********************
0246 0* ODFSND FILE
0247 0********************
0248 0*
0249 OODFSND  EADD                 ODFSRC
0250 0                             ODF#         4
0251 0                             OBJNAM      12
0252 0                             OBJTYP      13
0253 0                             SDATE       19
0254 0                             STIME       25
0255 0********************
0256 0* ODFMSG FILE
0257 0********************
0258 0*
0259 OODFMSG  EADD                 ODFTXT
0260 0                             ODF#         4
0261 0                             SYSTEM      12
0262 0                             OBJLIB      20
0263 0                             MSG1       260
0264 0                             MSG2       500
0265 0                             USER       508
```

Figure 14-21b

Screen format member ODFPOPFM

```
*. . . . 1 . . . . 2 . . . . 3 . . . . 4 . . . . 5 . . . . 6 . . . . 7 . . . . 8
     SODF       0124         Y                                   G
     D          32 225Y                            Y    CSEND OBJECTS THROUGH THX
     DE NETWORK
     DLIBRARY    8 270Y                   Y
     D          18 4 6Y                                 CSystem  (or blank)
     DSYSTEM     8 439Y  Y                           Y
     D          18 5 6Y                                 CUser ID  (or List)
     DUSER       8 539Y  Y                           Y
     D          26 6 6Y                                 CTime to Send Members    X
     D
     DDTIME      6 639Y  YD   Z                      Y
     D          69 8 6Y                            Y    C                        X
     D
     DOBJ1       810 6Y  Y                Y          Y
     DTYP1       11024Y  YA               Y          Y
     DDES1      401035Y  Y                           Y
     DOBJ2       811 6Y  Y                Y          Y
     DTYP2       11124Y  YA               Y          Y
     DDES2      401135Y  Y                           Y
     DOBJ3       812 6Y  Y                Y          Y
     DTYP3       11224Y  YA               Y          Y
     DDES3      401235Y  Y                           Y
     DOBJ4       813 6Y  Y                Y          Y
     DTYP4       11324Y  YA               Y          Y
     DDES4      401335Y  Y                           Y
     DOBJ5       814 6Y  Y                Y          Y
     DTYP5       11424Y  YA               Y          Y
     DDES5      401435Y  Y                           Y
     DOBJ6       815 6Y  Y                Y          Y
     DTYP6       11524Y  YA               Y          Y
```

```
DDES6      401535Y  Y                Y
DOBJ7       816 6Y  Y          Y     Y
DTYP7       11624Y  YA         Y     Y
DDES7      401635Y  Y                Y
DOBJ8       817 6Y  Y          Y     Y
DTYP8       11724Y  YA         Y     Y
DDES8      401735Y  Y                Y
DOBJ9       818 6Y  Y          Y     Y
DTYP9       11824Y  YA         Y     Y
DDES9      401835Y  Y                Y
DOBJ10      819 6Y  Y          Y     Y
DTYP10      11924Y  YA         Y     Y
DDES10     401935Y  Y                Y
DOBJ11      820 6Y  Y          Y     Y
DTYP11      12024Y  YA         Y     Y
DDES11     402035Y  Y                Y
DOBJ12      821 6Y  Y          Y     Y
DTYP12      12124Y  YA         Y     Y
DDES12     402135Y  Y                Y
D           923 6Y                          CMD 7-End
```

Figure 14-21a shows the ODFPOP workstation program, and Figure 14-21b shows the corresponding screen format member ODFPOPFM. Program ODFPOP first initializes some of the program variables and provides a default recipient of OPERATOR for our distributions. Next, program ODFPOP reads through the values that POP's LIBR procedure previously stored in the LDA. The ODFPOP program then loops through the array of 12 POP opcodes and their associated member names and types. When program ODF-POP finds an O opcode, it moves the associated member name and type into a new array to be displayed on the screen. The program then blanks out their former positions in the LDA and replaces the opcode O with an *. This section of code lets the utility deal with mixed opcodes and prevents the LIBRO procedure from being called more than once in a given execution of the LIBR# procedure. All the O opcodes are processed together, and all the associated data is cleared from the LDA so the LIBRO procedure is not evoked

Figure 14-22

Send Objects Through the Network screen

```
                        SEND OBJECTS THROUGH THE NETWORK
#POPLIB

System  (or blank)                    PORTLAND
User ID  (or List)                    OPERATOR
Time to Send Members                  000000

______________________________________________________________

ODFGET          S          SENDING THE SOURCE MEMBERS FOR THE
ODFPOP          S          POP-ODF UTILITY
ODFPOPFM        S          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________
________        _          ___________________________________

CMD 7-End
```

again. Because the LIBR# procedure is terminated at the first blank code, the * code keeps the LIBR# procedure from recalling the base POP LIBR procedure and effectively being terminated. When the LIBR# procedure finds the *, it looks for a procedure named LIBR*. Because the LIBR* procedure is merely a placeholder, it consists only of a // RETURN statement.

After reading the LDA array and filling the screen arrays, program ODFPOP outputs the screen (Figure 14-22). Pressing the Enter key opens the program, which chains to the direct file ODF.NUM to retrieve the last ODF transmission number used. The four-digit ODF transmission number provides a unique identification for each maintenance distribution. As soon as the ODF number is retrieved, it is incremented and written to file ODF.NUM. Next, the screen entries are written to disk. All the selected members and their types are written to the ODFSND file, while the message is written to the ODFMSG file. After the data has been written to the files, the program sets on LR, and control returns to the LIBRO procedure.

Figure 14-23a

Record layout of ODFSND file

```
*         1         2         3         4         5         6         7         8
0001 FODFSND  IP  F  32  32  4AI       1 DISK
0002 F*
0003 IODFSND  NS  98                                ODFSRC
0004 I                                          1   4 ODF#
0005 I                                          5  12 OBJNAM
0006 I                                         13  13 OBJTYP
0007 I                                         14  190SDATE
0008 I                                         20  250STIME
```

Figure 14-23b

Record layout of ODFMSG file

```
*         1         2         3         4         5         6         7         8
0001 FODFMSG  IP  F 512 512  4AI       1 DISK
0002 F*
0003 IODFMSG  NS  98                                ODFTXT
0004 I                                          1   4 ODF#
0005 I                                          5  12 SYSTEM
0006 I                                         13  20 OBJLIB
0007 I                                         21 260 MSG1
0008 I                                        261 500 MSG2
0009 I                                        501 508 USER
```

Figures 14-23a and 14-23b show the record layouts of the two files. Although I could have used one file with two arrays of 12 elements each for the object name and type, I chose to use two files. The two-file arrangement is more flexible than using one file with repeating data groups. In the future, I want to convert this utility to run from the Programming Development Manager (PDM) on the AS/400, and I don't want to be restricted to the 12-element limitation POP imposes.

Procedure LIBRO then checks for external switch U8. Command key 7 cancels the program and sets on external indicator U8 to abort any subsequent transmission. If you exited program ODFPOP using Command key 7, indicator U8 is set on, and procedure SENDODF is not evoked. If you exited program ODFPOP by pressing the Enter key, however, indicator U8 is set off, and procedure SENDODF is evoked.

The SENDODF Procedure

In the SENDODF procedure, the ODF services perform the actual distribution. As illustrated in Figure 14-24, the procedure checks the LDA in position 301 to determine whether a distribution time has been specified. If you entered a time, the procedure waits at the // WAIT statement until the specified time. If you didn't enter a time, the procedure continues processing, and the object distribution begins immediately. The ODFGET program is called to retrieve the library member names stored in the ODFSND file as well as the transmission message from the ODFMSG file.

Figure 14-24

Procedure SENDODF

```
// IFF ?L'301,6'?/000000 WAIT TIME-?L'301,6'?
// LOAD ODFGET
// FILE NAME-ODFSND,LABEL-ODF.SND,DISP-SHRRM
// FILE NAME-ODFMSG,LABEL-ODF.MSG,DISP-SHRRM
// PRINTER NAME-PRINTER,DEVICE-XP,FORMSNO-?L'307,4'?,PRIORITY-0
// RUN
// EVALUATE P10-?L'401,8'?
// EVALUATE P12-?L'409,8'?
SENDPRT F?L'307,4'?,CANCEL,?10?,?12?.....ACK
// EVALUATE P52-52
// EVALUATE P60-60
// TAG LOOP
// IF ?L'?52?,8'?/          CANCEL
// EVALUATE P22-'LIBRARY'
// IF ?L'60,1'?/O     EVALUATE P22-'LOAD'
// IF ?L'60,1'?/S     EVALUATE P22-'SOURCE'
// IF ?L'60,1'?/P     EVALUATE P22-'PROC'
SENDLIBR ?L'?52?,8'?,?22?,?L'1,8'?,,?10?,?12?....ACK
// EVALUATE P52-?52?+10
// EVALUATE P60-?60?+10
// GOTO LOOP
```

Program ODFGET (Figure 14-25) begins by setting up the initial values used in the program. The LDA of procedure SENDODF passes the ODF transmission number into the ODF# field. Remember, the LDA of an evoked job is copied from the evoking procedure. In this case, the LDA used in the original LIBRO procedure is carried into the SENDODF procedure and subsequently made available to program ODFGET. Next, program ODFGET retrieves the distribution data from the files and writes it to the LDA. The SYSTEM field is then checked for a blank entry. If the system entry is blank, the ODFGET program assumes that a list name is being used and moves the USER field that contains the list name into the DEST field, which contains the transmission destination. The report heading is then printed, and the ODF number gets the distribution members out of the ODFSND file, writes them back into the LDA, and prints them on the distribution report.

Figure 14-25

Program ODFGET

```
*         1         2       . 3         4         5         6         7         8
H     64                        8                                            ODFGET
F**************************************************************
F*------------PROGRAM DESCRIPTION-------------
F*
F* THIS PROGRAM UPOATES THE FIELDS OF A MACHINE RECORD.
F*
F**************************************************************
F*------------MAINTENANCE SUMMARY-------------
```

```
F*
F*  04/13/89 - MJO - PROGRAM WRITTEN
F*
F*****************************************************************
F*-------------FILE DESCRIPTION------------
F*
FODFSND  IF  F  32  32L 4AI     1 DISK
FODFMSG  IF  F 512 512  4AI     1 DISK
FPRINTER O   F 132 132     OF     PRINTER
FPRINTER1O   F 132 132     OA     PRINTER
F*
F*****************************************************************
F*------------INDICATOR SUMMARY-------------
F*
F********* INDICATORS **********
F*
F*     01  -  INPUT SCREEN1
F*     11  -  REUSABLE INDICATOR
F*
F*     97  -  READ WORKSTATION END OF FILE
F*
I*****************************************************************
I*------------ARRAYS AND TABLES  -------------
E                    POP        12  1                POP CODE
E                    OBJ        12  8                OBJECT NAME
E                    TYP        12  1                OBJECT TYPE
E                    MSG        12 40                MSG ARRAY
E                    OT         12 10                CODE(1)-OBJ(10)-TYP(1)
I*****************************************************************
I*------------FILE SPECIFICATIONS------------
I*
IODFSND  NS  98
I                                        1   4 ODF#
I                                        5  12 OBJNAM
I                                       13  13 OBJTYP
I                                       14  190SDATE
I                                       20  250STIME
F*
IODFMSG  NS  98
I                                        1   4 ODF#
I                                        5  12 SYSTEM
I                                       13  20 OBJLIB
I                                       21 260 MSG1
I                                      261 500 MSG2
I                                      501 508 USER
IOTDS        DS
I                                        1  10 COT
I                                        1   1 C
I                                        2   9 OBJNAM
I                                       10  10 OBJTYP
F* POP LDA SPECIFICATIONS
IPOPDS       UDS
I                                        1   8 LIB
I                                       51 170 OT
I                                      301 30600TIME
I                                      307 3100HOL040
I                                      401 408 USER
I                                      409 416 SYSTEM
I*
C*****************************************************************
C*----------CALCULATION SPECIFICATIONS----------
C*
C           ONCE      DO   0         ONCE    10
C                     MOVE 'N'       EOF     1
C                     MOVE 'N'       FOUNO   1
C                     MOVE HOLD40    ODF#
C                     MOVE *BLANK    OT
C                     TIME           UTIME   60
C                     END
C*
C*
C***-------------------------***
C*
C          ODF#       CHAINODFMSG                11
C   11                MOVE 'Y'       EOF
C*
```

```
C           EOF       IFNE 'Y'
C                     MOVEAMSG1      MSG,1
C                     MOVEAMSG2      MSG,7
C*
C           SYSTEM    IFEQ *BLANK
C                     MOVE USER      DEST    8
C                     ELSE
C                     MOVE SYSTEM    DEST
C                     END
C*
C                     EXCPTPRTREC
C*
C                     Z-ADD01        X       20
C           ODF#      SETLLODFSND
C           FOUND     DOUEQ'N'
C           ODF#      READEODFSND                   97
C  N97                MOVE 'Y'       FOUND
C   97                MOVE 'N'       FOUND
C           FOUND     IFEQ 'Y'
C                     MOVE OBJNAM    OBJ,X
C                     MOVE OBJTYP    TYP,X
C                     MOVE COT       OT,X
C                     ADD  1         X
C                     END
C                     END
C                     DO   12        X
C                     EXCPTPRTDTL
C                     END
C                     END
C*
C                     SETON                     LR
C*
C*
C***-------------------------***
C*
C*          ***       ***
C           END       TAG
C*          ***       ***
C*
C***-------------------------***
C*
C*
O*****************************************************************
O*----------FILE OUTPUT SPECIFICATIONS-----------
O*
O********************
O* PRINTER OUTPUT
O********************
O*
OPRINTER E  102           PRTREC
O                         ODF#        7
O                                    52 'ODF MAINTENANCE REQUEST'
O                                    70 'DATE:'
O                         UDATE Y    80
O*
O        E  1             PRTREC
O                         DEST       47
O                                    70 'TIME:'
O                         UTIME      80 '  :  :  '
O*
O        E  1             PRTREC
O                                    70 'PAGE:'
O                         PAGE       80
O*
O        E  2             PRTREC
O                         LIB        10
O*
O        E  1             PRTDTL
O                         OBJ,X      10
O                         TYP,X      20
O                         MSG,X      70
O********************
O* PRINTER OUTPUT
O********************
O*
OPRINTER1E  102           PRTREC
```

```
O                        ODF#        7
O                                   52 'ODF MAINTENANCE REQUEST'
O                                   70 'DATE-'
O                        UDATE Y    80
O*
O         E  1           PRTREC
O                        DEST       47
O                                   70 'TIME-'
O                        UTIME      80 '  .  '
O*
O         E  1           PRTREC
O                                   70 'PAGE-'
O                        PAGE       80
O*
O         E  2           PRTREC
O                        LIB        10
O*
O         E  1           PRTDTL
O                        OBJ,X      10
O                        TYP,X      20
O                        MSG,X      70
```

After the ODFGET program has loaded the library member names back into the LDA, the SENDODF procedure resumes by loading parameters 10 and 12 with the user and remote location names that will receive the transmission. Then, the ODF/36 SENDPRT procedure is called to send to the remote location the print spool file with the forms number that matches the ODF distribution number. Depending on the values specified in the ODF defaults at the remote site, the print spool file is either printed or held in the arrived objects folder at the remote location. Next, parameters 52 and 60 are loaded with the literal values of 52 and 60, respectively. Parameter 52 indicates the beginning LDA location of the member name array, while parameter 60 indicates the beginning LDA position of the member type array. The parameter is then substituted into the position portion of an LDA substitution statement to construct a moving pointer. If the member name position indicated by the pointer is blank, the SENDODF procedure is canceled. As long as the LDA position contains a value, the SENDODF procedure loops through the LDA array. Each loop executes the ODF/36 SENDLIBR procedure; parameters 52 and 60 are each incremented by 10 positions to provide pointers to the next possible LDA array locations.

Armed with an understanding of how to implement the POP and ODF/36 interface, you can now see how to implement this utility as a whole. Figure 14-26 shows a sample POP library members display where, using the O opcode, I've selected all our program source members. The ODF/36 POP interface lets you view the objects selected. You can optionally add members or make changes to the existing selections. Pressing Enter writes the screen data to the files and then evokes the SENDODF procedure to distribute the group of library members.

Although I don't cover the technique in this article, you could also customize the screen headings of your POP library members display to reflect your new O opcode. POP provides the LIBRCUST procedure to assist you in customizing your implementation. You can find more information in the POP on-line tutorial.

Figure 14-26

Sample POP library members display

```
Libr #POPLIB   Free sectors 1319/1606 of 5000       Free entries    47 of 244
 Operation codes  B Browse  S SEU      E Edit     O Delete   Y.Copy    N.Rename
 P Print  F SDA   K Backup  J Restore  C Compile  X Execute  H.History
 Command keys   1234 Column   5:Select   6 Library   7 End   11 Auto browse
 12 Condense    19 Off        HELP
 ______________________________________________________________________________

 _ ________ _         _ ________ _         _ ________ _         _ ________ _
 O ODFGET   S 164     _                    _                    _
 O ODFMSG   S 9       _                    _                    _
 O ODFPOP   S 223     _                    _                    _
 O ODFPOPFM S 50      _                    _                    _
 O ODFSND   S 8       _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
 _                    _                    _                    _
```

ODF/36 provides a flexible and reliable engine to perform the maintenance of your network. As delivered, the ODF/36 SENDLIBR procedure is primitive to operate, but by taking advantage of the user-extendable features found in both ODF/36 and POP, you can create an elegant and powerful tool to maintain the remote application libraries in your network. You can also adapt this utility to the POP files screen with slight modifications. All the concepts we discussed about the POP library members display are applicable to the POP files screen.

Putting a Job on the Job Queue from POP

by Noaman Afzal

Code on diskette:
Procedure LIBRQ2

To send a batch procedure directly from the POP library member display to the JOBQ, add the following LIBRQ procedure to #POPLIB:

```
// LIBRARY NAME-?3?
// JOBQ 3,?3?,?1?
```

Then key the Q operation code next to any procedure name to place that procedure on the job queue. After the selected procedures have been enqueued, you can resume your work with the library members display.

Note: Procedure LIBRQ is named LIBRQ2 on diskette. To use it in #POPLIB, you must rename it to LIBRQ. Another procedure described in Blanking Out Columns 1-5 and 75-80 in RPG Source with POP, *page 413, uses the name LIBRQ, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.*

Evoking a Job from POP

by Esteban Rivera and Matthew Henry

Code on diskette:
Procedure LIBRV

IBM's POP truly is a productivity aid, but when you execute a procedure via the X operation, you have to wait until the procedure finishes executing before you can be productive again. Adding the following LIBRV procedure to #POPLIB can solve this problem:

```
// LIBRARY NAME-?3?
// EVOKE ?1?
```

Procedure LIBRV adds to POP a new code, V, that allows you to EVOKE the selected procedure, leaving your terminal free for additional work. Note that procedure LIBRV changes the current library to the library name retrieved from positions 1 through 8 of the LDA and retrieves the procedure name from LDA positions 52 through 59, the positions POP normally assigns to these values.

Improving POP's File Copy

by Carl W. Selley

Two small changes to procedure FILEY in library #POPLIB will improve your S/36 POP file copy utility (Figure 14-27):

• Inserting a // REGION SIZE-64 statement before the // LOAD $COPY statement improves runtime.

• Adding the clause DISP-SHRRR to the FILE statement for the input file (COPYIN) lets you copy the input file while other users are reading it (but not updating the file).

Figure 14-27 *Modification to procedure FILEY*

```
* WILL COPY EVEN THOUGH OTHER JOBS ARE READING IT  NOT IF THEY ARE UPDATING
// PROMPT FORMAT-FILEY,MEMBER-LIBR@PRO  ?3'?1?'?
// IF ?CD?/2007 RETURN
// LIBRARY NAME-O
// MEMBER USER1-##MSG2
// REGION SIZE-64                                  <- New line
// LOAD $COPY
// FILE NAME-COPYIN,
*/ IFF ?2?/ DATE-?2?
// LABEL-?1?,DISP-SHRRR                            <- Modified line
```

```
// FILE NAME-COPYO,LABEL-?3?,DISP-NEW
// RUN
// COPYFILE OUTPUT-DISK,
// IF ?5?/ IFF ?4?/Y REORG-NO
// IF ?5?/ IF ?4?/Y DELETE-SYSDEL,REORG-YES
// IFF ?5?/ DELETE-'?6?,?5?',REORG-YES
// END
```

Renaming Single Files in POP

by John Cirocco

Code on diskette:
Procedure FILEN
Screen format member FILENFM

POP procedure command FILEN overcomes the shortcomings inherent in IBM's file RENAME procedure.

As many an experienced user has discovered, IBM's RENAME procedure leaves something to be desired. Before you can use the IBM procedure, you must know the exact spelling of the file to be renamed, as well as whether the new file name already is in use. And with the IBM procedure, you must type RENAME commands manually. POP users will find procedure command FILEN a convenient alternative to the IBM RENAME procedure.

Procedure command FILEN consists of procedure FILEN, a screen format member, and a prompt screen (Figure 14-28). With procedure FILEN (Figure 14-29), the user renames a file on disk via the custom POP operation code N. (POP gives you a list of files; you designate the file to be changed with the N operation code; POP automatically moves the the designated file name to the new name slot.) After verifying that a label exists on disk for the file selected, procedure FILEN employs screen format member FILENFM (Figure 14-30) to produce the prompt screen. The prompt screen displays the current file label, as well as an input field where the user can enter a new file label name.

Figure 14-28

Sample prompt screen for FILEN

```
FILENFM-FILEN            File Rename Procedure


                          Old File Name - ________


                          New File Name - ________

                This file name already exists - press CMD/7 to Cancel
                              or enter a  new File name
                             You must enter a file name


                             Press ENTER To Rename File

                                         or

                      Any Command Key to Cancel RENAME Procedure
```

With FILEN, you needn't remember the exact spelling of every file name in use on your system; you need only recognize the file name on POP's list of files. You also can use this list to ascertain the uniqueness of the new file name you are considering. Once you enter the new name, FILEN double-checks to ensure that the new name indeed is unique and that the new file name field was not inadvertently blank. If FILEN finds no problem, the name change is a *fait accompli*. But if the new name already is in use or the field is blank, you receive an error message. In either error situation, you have the option of retrying or canceling.

In addition to reducing reliance on human memory and eliminating error situations, FILEN expedites the name change process by reducing the amount of typing involved. IBM's RENAME procedure requires retyping the entire file name; procedure command FILEN automatically places the name of the file selected for renaming into both the current and new name fields on the prompt screen, and the default for the new file name is the current label name. So with FILEN, you simply can elect to modify the old file name that automatically appears in the new file name slot. This approach is advantageous if the new file name differs only slightly from an existing file name — for example, if only the file group is to be changed or if only a dot is to be added or removed.

So if file name changes in your shop take longer than you would like, or if the name change procedure all too often results in name duplication errors, give FILEN a try.

Figure 14-29

Procedure FILEN

```
*******************************************************************
**                     11-10-87  JOHN W  CIROCCO                 **
**                        EASTMAN KOOAK/WWBIS                    **
**                         Proc Name  FILEN                      **
*******************************************************************
*
** MODIFICATION TO POP - NEW PROCEDURE - FILE RENAME WITH AUDITS
** P1-FILE TO 8E RENAMED (TAKEN FROM POP'S FILE SCREEN)
** P61-FILE TO 8E RENAMED
** P62-NEW NAME OF FILE (DEFAULTS TO P1 FOR EASIER CHANGE)
** UPSI SWITCH 1 + 2 - NON-DISPLAY OF ERROR MESSAGES DN PROMPT SCREEN
** UPSI SWITCH 8 - SOUND ALARM ON ERROR
*
// IFF DATAF1-?1?                 RETURN
// EVALUATE P61-'?1?'
// EVALUATE P82-'?1?'
// SWITCH 11000000
// INFOMSG NO
// TAG TOP
// PRDMPT MEM8ER-FILENFM,FORMAT-FILEN,LENGTH-'8 8',START-61,UPSI-YES
// IFF ?CD?/0000        RETURN
// SWITCH 11000000
// IF ?62?/                        SWITCH XOXXXXX1
// IF SWITCH8-1                   GOTO TDP
// IF DATAF1-?62?                  SWITCH OXXXXXX1
// IF SWITCH8-1                   GOTD TDP
// RENAME ?61?,?62?
// INFOMSG YES
```

Figure 14-30

Screen format member FILENFM

```
*...  .. 1 ... ... 2 ... . . 3 ... ... 4 ... ... 5 ... ... 6 . . ... 7 .. .. 8
S*************************************************************************
S**                     11-10-87  JOHN W. CIROCCO                       **
S**                        EASTMAN KODAK/WWBIS                          **
S**                         Proc Name: FILEN                            **
S*************************************************************************
S*
S** MODIFICATION TO POP - SCREEN FORMAT - FILE RENAME WITH AUDITS
S** P61-FILE TO BE RENAMED
S** P62-NEW NAME OF FILE (DEFAULTS TO P61 FOR EASIER CHANGE)
S** INDICATOR 91 + 92 - NON-DISPLAY OF ERROR MESSAGES ON PROMPT SCREEN
S** INDICATOR 98 - SOUND ALARM ON ERROR
S*
SFILEN            98YN        Y
D          13 1 5Y                                CFILENFM-FILEN
D          21 130Y                                CFile Rename Procedure
D          15 528Y              Y                 COld File Name -
DPARA61     8 544Y  YB        Y Y
D          15 928Y              Y                 CNew File Name -
DPARA62     8 944Y  YB          Y         Y
DERR91A    531114Y                   91Y          CThis file name already X
Dexists - press CMD/7 to Cancel
DERR91B    251228Y                   91Y          Cor enter a  new File naX
Dme
DERR92     261327Y                   92Y          CYou must enter a file nX
Dame
D          261627Y                                CPress ENTER To Rename FX
Dile
D           21839Y                                Cor
D          422019Y                                CAny Command Key to CancX
Del RENAME Procedure
```

Renaming and Copying Multiple Files in POP

by Tim Hack

Code on diskette:

Procedures FILEQ, FQQ, FILES, FSQ, FILVPARM
Screen format members FILEQQFM, FILESSFM

Regular IBM procedures let you rename or copy only one file at a time. But with procedure commands FIlEQ and FILES, you can EVOKE or place up to 12 rename or copy requests on the JOBQ at a time.

When you rename a file, you often copy it. When you copy a file, you often rename it. With the IBM RENAME and COPYDATA procedures, you can rename or copy only one file at a time. With POP procedure commands FILEQ and FILES, though, you can copy or rename several files at once using prompt screens like those in Figures 14-31 and 14-32.

Procedure command FILEQ lets you queue up to 12 files for renaming. If a particular rename function must be aborted because the new name is not unique, FILEQ returns a message to the user at runtime. FILEQ likewise returns a message to the user at runtime if a renaming function could not be attempted because the new file name field was inadvertently blank. And procedure command FILEQ also accommodates date-differentiated files. Its primary advantage, though, is the ability to EVOKE a "batch" of renames or to place the batch on the JOBQ.

Procedure command FILES, which invokes the COPYDATA procedure, is similar to FILEQ and, in fact, includes all FILEQ benefits. Particularly useful when you must copy many files for testing, procedure command FILES is easier and faster than keying in 12 // EVOKE COPYDATA statements with the DUP key.

Figure 14-31
Sample prompt screen for FILEQ

```
FILEQQAA        QUICK RENAME UTILITY USING POP FILE UTILITY

Old File Name/Date          New File Name

00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000



Press <ENTER> to RUN Request                          CMD 5 to EVOKE
Request
CMD 4 to JOBQ Request                                 CMD 7 to CANCEL
Request
```

Figure 14-32
Sample prompt screen for FILES

```
FILESSAA        QUICK FILE COPY UTILITY USING POP FILE UTILITY

Copy From                   Copy To
File Name  Date             File Name

00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000
00000000  000000  ----> 00000000

NOTE: If 1) Copy To File Name already EXISTS on disk  2) Copy To File Name
is
BLANK  3) Copy To File Name is the same as Copy From File Name
4) Copy From File Name does NOT exist  then file will NOT be copied.

Press <ENTER> to RUN Request                          CMD 5 to EVOKE
Request
CMD 4 to JOBQ Request                                 CMD 7 to CANCEL
Request
```

Procedure command FILEQ consists of a procedure and a screen format member. Screen format member FILEQQFM (Figure 14-33) produces prompt screen FILEQQAA (Figure 14-31). Procedure FILEQ (Figure 14-34) is used if the job is being EVOKEd or run from the terminal; procedure FQQ (Figure 14-35) is used if the job is being run from the JOBQ.

Utility FILES similarly consists of a procedure and screen format member. Screen format member FILESSFM (Figure 14-36) produces prompt screen FILESSAA (Figure 14-32). Procedure FILES (Figure 14-37) is used if the job is being EVOKEd or run from the terminal; procedure FSQ (Figure 14-38) is used if the job is being run from the JOBQ. To access each subsequent parameter (i.e., each subsequent file for renaming or copying), both procedure command FILEQ and procedure command FILES call the same subprocedure: FILVPARM (Figure 14-39).

The Q and S opcodes can be used in conjunction with each other (within one group of opcode requests) and with all standard POP file opcodes. The queuing maximum of 12 is based on POP's own file opcode limitation.

FILEQ and FILES accumulate all their respective requests (each file selected by a Q or S opcode) and display the collected file names and dates on a single screen. An input field next to each file name lets you enter output file names. The initial Q or S screen defaults each output file name to the original input file name with null fill capability (to allow keyboard insert), thereby letting the user alter input file names quickly to new output file names.

The standard LDA positions (001-200) used to hold the file name and date information within POP are used but not altered by these opcodes. LDA positions 507 through 511 are reserved to control the execution of the FILEQ opcode, and LDA positions 502 through 506 are reserved to control the execution of the FILES opcode. These reserved LDA positions *cannot* be used for any other purpose while POP is in use in FILE mode. If abnormal termination of POP's FILE mode occurs during FILEQ or FILES execution, you should clear LDA positions 507 through 511 or 502 through 506 respectively to reset control and allow future use of these opcodes during the current workstation session.

Both display screens FILEQQAA and FILESSAA provide CANCEL, EVOKE, JOBQ, and LOCAL RUN execution modes for the queued requests. If the JOBQ or EVOKE options are selected (via a command key) and if opcode execution fails, an informational message is sent to the requesting user about failure on a file-by-file basis.

The FILEQ and FILES procedure commands save a tremendous amount of time in shops that copy and rename large numbers of files.

Figure 14-33

Screen format member FILEQQFM

```
*     ... 1 ..    .. 2 ... 3 ... ... 4 ... ... 5 ... ..  6 ... ... 7 ... ... 8
SFILEQQAA             Y        Y                          DEG
D           8 2 3Y                                CFILEQQAA
D          43 220Y               Y                CQUICK RENAME UTILITY USX
DING POP FILE UTILITY
D          18 518Y               Y       Y        COld File Name/Date
D          13 546Y               Y       Y        CNew File Name
DFILI01     8 720Y  Y          Y Y   51
DDATE01     6 731Y  Y          Y Y   51
D           8 739Y               Y   51           C---->
DFIL001     8 748Y  Y        N 51Y   51
DFILI02     8 820Y  Y          Y     52
DDATE02     6 831Y  Y          Y     52
D           8 839Y                   52           C---->
DFIL002     8 848Y  Y        N 52    52
DFILI03     8 920Y  Y          Y Y   53
DDATE03     6 931Y  Y          Y Y   53
D           8 939Y               Y   53           C---->
```

```
DFIL003    8 948Y  Y        N 53Y   53
DFILI04    81020Y  Y          Y     54
DDATE04    61031Y  Y          Y     54
D          81039Y                   54         C---->
DFIL004    81048Y  Y        N 54    54
DFILI05    81120Y  Y          Y Y   55
DDATE05    61131Y  Y          Y Y   55
D          81139Y               Y   55         C---->
DFIL005    81148Y  Y        N 55Y   55
DFILI06    81220Y  Y          Y     56
DDATE06    61231Y  Y          Y     56
D          81239Y                   56         C---->
DFIL006    81248Y  Y        N 56    56
DFILI07    81320Y  Y          Y Y   57
DDATE07    61331Y  Y          Y Y   57
D          81339Y               Y   57         C---->
DFIL007    81348Y  Y        N 57Y   57
DFILI08    81420Y  Y          Y     58
DDATE08    61431Y  Y          Y     58
D          81439Y                   58         C---->
DFIL008    81448Y  Y        N 58    58
DFILI09    81520Y  Y          Y Y   59
DDATE09    61531Y  Y          Y Y   59
D          81539Y               Y   59         C---->
DFIL009    81548Y  Y        N 59Y   59
DFILI10    81620Y  Y          Y     60
DDATE10    61631Y  Y          Y     60
D          81639Y                   60         C---->
DFIL010    81648Y  Y        N 60    60
DFILI11    81720Y  Y          Y Y   61
DDATE11    61731Y  Y          Y Y   61
D          81739Y               Y   61         C---->
DFIL011    81748Y  Y        N 61Y   61
DFILI12    81820Y  Y          Y     62
DDATE12    61831Y  Y          Y     62
D          81839Y                   62         C---->
DFIL012    81848Y  Y        Y 62    62
D          2823 2Y              Y              CPress <ENTER> to RUN ReX
Dquest
D          222356Y                             CCMD 5 to EVOKE Request
D          2124 2Y                             CCMD 4 to JOBQ Request
D          232456Y              Y              CCMD 7 to CANCEL Request
```

Figure 14-34

Procedure FILEQ

```
**********************************************************************
**                       03-25-88  TIM A. HACK                      **
**                       EASTMAN KODAK/WWBIS                        **
**                        Proc Name: FILEQ                          **
**********************************************************************
// IF EVOKED-YES                       GOTO EVOKRUN
// IFF ?L'511,1'?/Q                    GOTO 1STPAS
**
// EVALUATE P45,2=?L'509,2'?+1
// LOCAL OFFSET-509,DATA-'?45?'        SAVE CURRENT FILE REQUEST COUNT
// IF ?L'507,2'?/?L'509,2'?            LOCAL OFFSET-507,DATA-'    '
// RETURN
**
// TAG 1STPAS
**
** INITIALIZE VALUES FOR VARIABLE PARM LOADING
**
// IF ?L'511,1'?/Q                     GOTO SKIP90
// LOCAL OFFSET-507,DATA-'0000'        ZERO OUT LDA CTRL COUNTS AT 1ST PASS
// EVALUATE P44,2=00
// EVALUATE P40,3=021
// EVALUATE P41,2=00
// EVALUATE P51='X'  P52='X'  P53='X'  P54='X'  P55='X'  P56='X'
// EVALUATE P57='X'  P58='X'  P59='X'  P60='X'  P61='X'  P62='X'
// EVALUATE P64='Q'
**
** CALL PROC TO ASSIGN VALUES TO VARIABLE PARMS
**
// INCLUDE FILVPARM *ALL
// LOCAL OFFSET-507,DATA-'?44?'        SAVE HOW MANY FILEQ REQUESTS FOUND
```

```
// LOCAL OFFSET-511,DATA-'Q'              ONE TIME PROCESS CONTROL UPDATE TO LDA
// EVALUATE P45,2=?L'509,2'?+1            INCREMENT NTH PASS FOR FILEQ COUNTER
// LOCAL OFFSET-509,DATA-'?45?'           SAVE NTH PASS FOR FILES COUNTER
**
** PROMPT SCREEN LOAD FOR ENTERING OUTPUT RENAME FILE NAMES
**
// PROMPT MEMBER-FILEQQFM,FORMAT-FILEQQAA,
// LENGTH-'8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8'
// IF ?CD?/2007      IF ?L'507,2'?/?L'509,2'?       LOCAL OFFSET-507,DATA-'      '
// IF ?CD?/2007                           RETURN
**
// EVALUATE P50=''
// IF ?CD?/2004                           EVALUATE P50=J
// IF ?CD?/2005                           EVALUATE P50=E
// IF ?50?/E                              EVOKE FILEQ *ALL
// IF ?50?/E         IF ?L'507,2'?/?L'509,2'?       LOCAL OFFSET-507,DATA-'      '
// IF ?50?/E                              RETURN
**
// IFF ?50?/J                             GOTO EVOKRUN
// IFF PROC-'FQQ,?CLIB?'                  GOTO EVOKRUN
**
** SETUP TO RUN FROM JOBQ IF JOBQ PROC EXISTS.  ONLY PASS OUTPUT FILE PARMS.
**
// JOBQ ,FQQ,?03?,?06?,?09?,?12?,?15?,?18?,?21?,?24?,?27?,?30?,?33?,?36?
// IF ?50?/J        IF ?L'507,2'?/?L'509,2'?        LOCAL OFFSET-507,DATA-'      '
// IF ?50?/J                              RETURN
**
** REQUEST LOOP ROUTINE FOR EVOKED REQUEST
**
// TAG EVOKRUN
// IF EVOKED-NO                           INFOMSG NO
**
// EVALUATE P48,2=04
**
// IF ?50?/J        IFF JOBQ-YES          GOTO LOOPRN
// TAG RELOAD
// LOAD $RENAM
// RUN
**
** RENAME ALL FILES WITH $RENAM CALL USING VARIABLE PARM LOADING
**
// TAG LOOPRN
// EVALUATE CD=0000
// EVALUATE P48=?48?-1
// EVALUATE P49=?48?/3
// IF ?49?>?L'507,2'?                     GOTO ENDRNM
**
** LOAD VARIABLE PARM TO P39 WITH FILE NAME OUTPUT FOR $RENAM
** LOAD VARIABLE PARM TO P38 WITH FILE NAME INPUT DATE FOR $RENAM
** LOAD VARIABLE PARM TO P37 WITH FILE NAME INPUT FOR $RENAM
**
// EVALUATE P39=??48??
// EVALUATE P48=?48?-1
// EVALUATE P38,6=??48??
// EVALUATE P48=?48?-1
// EVALUATE P37=??48??
**
// IFF ?39?/     IF DATAF1-?39?                         EVALUATE CD=2030
// IFF ?38?/     IFF ?37?/     IFF DATAF1-'?37?,?38?'   EVALUATE CD=2030
// IF  ?38?/     IFF ?37?/     IFF DATAF1-?37?          EVALUATE CD=2030
// IF  ?CD?=2030                          END
// IFF ?50?/E                             GOTO NOEVMSG
// IF  ?CD?=2030 MSG ?USER?, FILE NAMED ?37? NOT RENAMED TO ?39? DURING EVOKED EXECUTE.
**
// TAG NOEVMSG
// IF  ?CD?=2030                          GOTO NORENM
**
// IFF ?38?/    IF DATAF1-'?37?,?38?'     IFF DATAF1-?39?+
   RENAME LABEL-?37?,NEWLABEL-?39?,DATE-?38?
// IF ?38?/     IF DATAF1-?37?            IFF DATAF1-?39?+
   RENAME LABEL-?37?,NEWLABEL-?39?
**
// TAG NORENM
// EVALUATE P48=?48?+6
// IF  ?CD?=2030                          GOTO RELOAD
```

```
// GOTO LOOPRN
// TAG ENDRNM
// END
**
// IF ?L'507,2'?/?L'509,2'?             LOCAL OFFSET-507,DATA-'
// IF EVOKED-NO                         INFOMSG YES
// RETURN
*********************************************************************
**                                                                 **
**           GROUP RENAME UTILITY USING POP FILE UTILITY           **
**                                                                 **
** USES "POP" DISPLAY FILES UTILITY WITH NEW "Q" CHARACTER AS OP CODE.  **
** FILES REQUESTED FOR "Q" RENAME WILL BE DISPLAYED ON PROMPT SCREEN    **
** WITH THE OLD FILE NAME & DATE & NEW FILE NAME (DEFAULTS TO OLD NAME). **
** NEW FILE NAME FIELD IS THE ONLY FIELD ALLOWED FOR INPUT BY USER.     **
** MAXIMUM OF 12 FILES (ONE SET) MAY BE SETUP AT ONE TIME FOR "Q" RENAME.**
**                                                                 **
** RECOVERY NOTE: IF USER INTERRUPTS/CANCELS "Q" RENAME REQUESTS, THEN  **
**                LDA POSITIONS 507 THRU 511 SHOULD BE SET TO BLANKS TO **
**                CONTINUE TO ALLOW USE OF QUICK RENAME UTILITY  THESE  **
**                POSITIONS ARE USED TO CONTROL ONE SET OF "Q" RENAMES  **
**                                                                 **
**      LDA NOTE  DO NOT RE-USE LDA POSITION 502 THRU 506 ALREADY USED  **
**                FOR FILES CONTROL   NEED TO KEEP POSITION 502 THRU 506 **
**                INTACT FOR LIFE OF FILE SET REQUEST WITHIN POP        **
**                                                                 **
**                                                                 **
** PROCEDURES CALLED                                               **
** -----------------                                               **
** FILVPARM - INITIAL LOAD FILE/DATE INPUT & FILE OUTPUT TO VARIABLE PARM**
** FQQ      - JOBQ RUN OF THIS PROCESS IF REQUESTED DURING PROMPT   **
**                                                                 **
** PARAMETER DEFINITIONS:                                          **
** ----------------                                                **
** P01 - P36  FILE NAME AND DATE PARAMETERS USED IN $RENAM PROCESS  **
** P01 - P12  REUSED AS NEW FILE NAME OUTPUT PARMS PASSED TO JOBQ CALL  **
** P37 - WORKING PARM FOR $RENAM FILE NAME INPUT                    **
** P38 - WORKING PARM FOR $RENAM FILE NAME INPUT DATE               **
** P39 - WORKING PARM FOR $RENAM FILE NAME OUTPUT                   **
** P40 - OP CODE POSITION IN LDA WHICH IS LOADED BY POP UTILITY     **
** P41 - NTH PARM CONTAINING INPUT FILE NAME                        **
** P42 - NTH PARM CONTAINING INPUT FILE DATE                        **
** P43 - NTH PARM CONTAINING NEW OUTPUT FILE NAME.                  **
** P44 - COUNTER FOR TOTAL NUMBER OF CURR FILE REQUEST THAT ARE "Q".  **
** P45 - NTH PASS FOR "Q" RENAME REQUEST WITHIN ONE FILE SET        **
** P46 - FILE NAME POSITION IN LDA WHICH IS LOADED BY POP UTILITY   **
** P47 - FILE DATE POSITION IN LDA WHICH IS LOADED BY POP UTILITY   **
** P48 - WORKING PARM TO CONTROL $RENAM PARM VARIABLE PARM LOADING  **
** P49 - WORKING PARM TO CONTROL $RENAM LOOPING                     **
** P50 - MEMORY FOR REQUEST FOR JOBQ OR EVOKED JOBS                 **
** P51 - P63  ATTRIBUTE CONTROL PARMS USED IN SCREEN PROMPT         **
** P64 - PROCESS CONTROL CHARACTER PASSED TO LOAD VARIABLE PARM PROC.  **
**                                                                 **
** LDA USAGE:                                                      **
** -----                                                           **
** 021 - 200  CURRENTLY USED BY FILE# PROC WITHIN POP AND ACCESSED HERE  **
** 507 - 508  STORES TOTAL NUMBER OF "Q" RENAME REQUEST FOR ONE FILE SET **
** 509 - 510  NTH PASS WITHIN ONE FILE SET FOR "Q" RENAME REQUEST    **
** 511 - 511  "Q" RENAME PROMPT & $RENAM EXECUTION CONTROL          **
**            ALL "Q" REQUESTS ARE IDENTIFIED AND SETUP FOR EXECUTION  **
**            DURING FIRST "Q" REQUEST FOUND IN FILE SET   ALL OTHER  **
**            ATTEMPTS TO RUN "Q" REQUEST WITHIN ONE FILE SET WILL NOT  **
**            EXECUTE PROMPT & $RENAM SINCE LDA 511 IS SET TO "D" AFTER  **
**            FIRST PASS THRU "Q" REQUEST IN ONE FILE SET.  AFTER ALL  **
**            "Q" REQUESTS HAVE PASSED LDA 507 THRU 511 IS BLANKED OUT.  **
*********************************************************************
**
**
** END OF FILEQ PROCEDURE
```

Note: Another procedure described in Improving and Adding Operations in POP, *page 449, uses the name FILEQ, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.*

Figure 14-35
Procedure FQQ

```
**********************************************************************
**                        03-24-88  TIM A. HACK                    **
**                         EASTMAN KODAK/WWBIS                     **
**                         Proc Name: FQQ  (JOBQ REQUEST FROM FILEQ) **
**********************************************************************
**
** REPOSITION NEW OUTPUT FILE NAMES PASSED AS PARM 01 - 12 TO ORIGINAL PARMS
// EVALUATE P41,2=12                    INITIAL INCOMING PARM ID SET TO 12
// EVALUATE P43,2=00
**
// TAG REPOS
// EVALUATE P43,2=?41?*3                REPOSITION INCOMING PARM VALUE TO THIS PARM
// IFF ??41??/                          EVALUATE P?43?=??41??
// ELSE                                 EVALUATE P?43?=''
// EVALUATE P41=?41?-1                  COUNTDOWN NEXT INCOMING PARM TO REPOS
// IF ?41?>00                           GOTO REPOS
**
** INITIALIZE VALUES FOR VARIABLE PARM LOADING IN FILVPARM PROC.
**
// LOCAL OFFSET-507,DATA-'0000'         ZERO OUT LDA CTRL COUNTS AT 1ST PASS
// EVALUATE P44,2=00
// EVALUATE P40,3=021
// EVALUATE P41,2=00
// EVALUATE P64='Q'
**
** CALL PROC TO ASSIGN VALUES TO VARIABLE PARMS
**
// INCLUDE FILVPARM *ALL
**
// LOCAL OFFSET-507,DATA-'?44?'         SAVE HOW MANY FILES REQUESTS FOUND
**
// EVALUATE P48,2=04
**
// TAG RELOAD
// LOAD $RENAM
// RUN
**
** RENAME ALL FILES WITH $RENAM CALL USING VARIABLE PARM LOADING
**
// TAG LOOPRN
// EVALUATE CD=0000
// EVALUATE P48=?48?-1
// EVALUATE P49=?48?/3
// IF ?49?>?L'507,2'?                   GOTO ENDRNM
**
** LOAD VARIABLE PARM TO P39 WITH FILE NAME OUTPUT FOR $RENAM
** "                 " TO P38 WITH FILE NAME INPUT DATE FOR $RENAM
** "                 " TO P37 WITH FILE NAME INPUT FOR $RENAM
**
// EVALUATE P39=??48??
**
// EVALUATE P48=?48?-1
// EVALUATE P38,6=??48??
**
// EVALUATE P48=?48?-1
// EVALUATE P37=??48??
// IFF ?39?/     IF DATAF1-?39?                          EVALUATE CD=2030
// IFF ?38?/     IFF ?37?/     IFF DATAF1-'?37?,?38?'    EVALUATE CD=2030
// IF  ?38?/     IFF ?37?/     IFF DATAF1-?37?           EVALUATE CD=2030
// IF  ?CD?=2030                        END
// IF  ?CD?=2030 MSG ?USER?, FILE NAMED ?37? NOT RENAMED TO ?39? DURING JOBQ EXECUTION
// IF  ?CD?=2030                        GOTO NORENM
**
// IFF ?38?/    IF DATAF1-'?37?,?38?'   IFF DATAF1-?39?+
   RENAME LABEL-?37?,NEWLABEL-?39?,DATE-?38?
// IF ?38?/     IF DATAF1-?37?          IFF DATAF1-?39?+
   RENAME LABEL-?37?,NEWLABEL-?39?
**
// TAG NORENM
// EVALUATE P48=?48?+6
// IF  ?CD?=2030                        GOTO RELOAD
// GOTO LOOPRN
// TAG ENDRNM
// END
**
```

```
// RETURN
***************************************************************************
**                                                                       **
**     MULTIPLE FILEQ RENAME USING POP FILE UTILITY JOBQ REQUEST        **
**                                                                       **
** THIS IS JOBQ PROC EXECUTED IF FILEQ USES JOBQ COMMAND KEY REQUEST.   **
** SEE FILEQ PROC FOR MORE COMPLETE DESCRIPTION.                         **
**                                                                       **
** PARAMETER DEFINITIONS:                                                **
** ------------                                                          **
** SEE FILEQ PROC DOCUMENTATION FOR PARM DEFINITIONS.                    **
**                                                                       **
** LDA USAGE:                                                            **
** -----                                                                 **
** SEE FILEQ PROC DOCUMENTATION FOR LDA MAPPING FOR THIS ENTIRE FUNCTION.**
***************************************************************************
**
** END OF FQQ PROCEDURE
```

Figure 14-36

Screen format member FILESSFM

```
*    ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
     SFILESSAA             Y        Y                          DEG
     D           8 1 3Y                                CFILESSAA
     D          46 120Y               Y                CQUICK FILE COPY UTILITYX
     D USING POP FILE UTILITY
     DFA0006     9 320Y               Y                CCopy From
     DFA0003     7 348Y               Y                CCopy To
     D           9 420Y               Y       Y        CFile Name
     DFA0002     4 431Y               Y       Y        CDate
     D           9 448Y               Y       Y        CFile Name
     DFILI01     8 620Y  Y          Y Y   51
     DDATE01     6 631Y  Y          Y Y   51
     D           8 639Y               Y   51           C---->
     DFIL001     8 648Y  Y        N 51Y   51
     DFILI02     8 720Y  Y          Y     52
     DDATE02     6 731Y  Y          Y     52
     D           8 739Y                   52           C---->
     DFIL002     8 748Y  Y        N 52    52
     DFILI03     8 820Y  Y          Y Y   53
     DDATE03     6 831Y  Y          Y Y   53
     D           8 839Y               Y   53           C---->
     DFIL003     8 848Y  Y        N 53Y   53
     DFILI04     8 920Y  Y          Y     54
     DDATE04     6 931Y  Y          Y     54
     D           8 939Y                   54           C---->
     DFIL004     8 948Y  Y        N 54    54
     DFILI05     81020Y  Y          Y Y   55
     DDATE05     61031Y  Y          Y Y   55
     D           81039Y               Y   55           C---->
     DFIL005     81048Y  Y        N 55Y   55
     DFILI06     81120Y  Y          Y     56
     DDATE06     61131Y  Y          Y     56
     D           81139Y                   56           C---->
     DFIL006     81148Y  Y        N 56    56
     DFILI07     81220Y  Y          Y Y   57
     DDATE07     61231Y  Y          Y Y   57
     D           81239Y               Y   57           C---->
     DFIL007     81248Y  Y        N 57Y   57
     DFILI08     81320Y  Y          Y     58
     DDATE08     61331Y  Y          Y     58
     D           81339Y                   58           C---->
     DFIL008     81348Y  Y        N 58    58
     DFILI09     81420Y  Y          Y Y   59
     DDATE09     61431Y  Y          Y Y   59
     D           81439Y               Y   59           C---->
     DFIL009     81448Y  Y        N 59Y   59
     DFILI10     81520Y  Y          Y     60
     DDATE10     61531Y  Y          Y     60
     D           81539Y                   60           C---->
     DFIL010     81548Y  Y        N 60    60
     DFILI11     81620Y  Y          Y Y   61
     DDATE11     61631Y  Y          Y Y   61
     D           81639Y               Y   61           C---->
     DFIL011     81648Y  Y        N 61Y   61
```

```
54 DFILI12      61720Y  Y          Y     62
55 DDATE12      61731Y  Y          Y     62
56 D            61739Y                   62          C---->
57 DFILO12      61748Y  Y        Y 62    62
56 D            519 4Y               Y               CNOTE:
59 D           2261910Y                              CIf 1) Copy To File NameX
60 D already EXISTS on disk  2) Copy To File Name is        BLANK  3) Copy TX
61 Do File Name is the same as Copy From File Name                 4) Copy FX
62 Drom File Name does NOT exist  then file will NOT be copied
63 D           2823 2Y               Y               CPress <ENTER> to RUN ReX
64 Dquest
65 D           222356Y                               CCMD 5 to EVOKE Request
66 D           2124 2Y                               CCMD 4 to JOBQ Request
67 D           232456Y               Y               CCMD 7 to CANCEL Request
```

Figure 14-37

Procedure FILES

```
*****************************************************************************
**                          03-12-88  TIM A  HACK                          **
**                          EASTMAN KODAK/WWBIS                            **
**                           Proc Name  FILES                              **
*****************************************************************************
// IF EVOKED-YES                          GOTO EVOKRUN
// IFF ?L'506,1'?/S                       GOTO 1STPAS
**
// EVALUATE P45,2=?L'504,2'?+1
// LOCAL OFFSET-504,DATA-'?45?'           SAVE CURRENT FILE REDUEST COUNT
// IF ?L'502,2'?/?L'504,2'?               LOCAL OFFSET-502,DATA-'    '
// RETURN
**
// TAG 1STPAS
**
** INITIALIZE VALUES FOR VARIABLE PARM LOAOING
**
// LOCAL OFFSET-502,OATA-'0000'           ZERO OUT LDA CTRL COUNTS AT 1ST PASS
// EVALUATE P44,2=00
// EVALUATE P40,3=021
// EVALUATE P41,2=00
// EVALUATE P51='X'  P52='X'  P53='X'  P54='X'  P55='X'  P56='X'
// EVALUATE P57='X'  P58='X'  P59='X'  P60='X'  P61='X'  P62='X'
// EVALUATE P64='S'
**
** CALL PROC TO ASSIGN VALUES TO VARIABLE PARMS
**
// INCLUOE FILVPARM *ALL
**
// LOCAL OFFSET-502,DATA-'?44?'           SAVE HOW MANY FILES REQUESTS FOUND
// LOCAL OFFSET-506,DATA-'S'              ONE TIME PROCESS CONTROL UPDATE TO LDA
// EVALUATE P45,2=?L'504,2'?+1            INCREMENT NTH PASS FOR FILES COUNTER
// LOCAL OFFSET-504,OATA-'?45?'           SAVE NTH PASS FOR FILES COUNTER
**
** PROMPT SCREEN LOAD FOR ENTERING OUTPUT COPY FILE NAMES
**
// PROMPT MEMBER-FILESSFM,FORMAT-FILESSAA,
// LENGTH-'8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8,8,6,8'
// IF ?CD?/2007      IF ?L'502,2'?/?L'504,2'?       LOCAL OFFSET-502,DATA-'    '
// IF ?CD?/2007                           RETURN
**
// EVALUATE P50-''
// IF ?CD?/2004                           EVALUATE P50=J
// IF ?CD?/2005                           EVALUATE P50=E
// IF ?50?/E                              EVOKE FILES *ALL
// IF ?50?/E        IF ?L'502,2'?/?L'504,2'?       LOCAL OFFSET-502,DATA-'    '
// IF ?50?/E                              RETURN
**
// IFF ?50?/J                             GOTO EVOKRUN
// IFF PROC-'FSO,?CLIB?'                  GOTO EVOKRUN
**
** SETUP TO RUN FROM JOBD IF JOBQ PROC EXISTS   ONLY PASS OUTPUT FILE PARMS
**
// JOBO ,FSD,?03?,?06?,?09?,?12?,?15?,?18?,?21?,?24?,?27?,?30?,?33?,?36?
// IF ?50?/J        IF ?L'502,2'?/?L'504,2'?       LOCAL OFFSET-502,OATA-'    '
```

```
// IF ?50?/J                            RETURN
**
// TAG EVOKRUN                          INITIAL PLACEMENT FOR EVOKED PROC
// IF EVOKED-NO                         INFOMSG NO
**
** COPY ALL QUEUED FILES FOR COPY WITH $COPY CALL WITH VARIABLE PARM LOADING
**
// EVALUATE P48,2-04
**
// TAG LOOPCP
// EVALUATE CD-0000
// EVALUATE P48-?48?-1
// EVALUATE P49-?48?/3
// IF ?49?>?L'502,2'?                   GOTO ENDCPY
**
** LOAD VARIABLE PARM TO P39 WITH FILE NAME OUTPUT FOR $COPY
** LOAD VARIABLE PARM TO P38 WITH FILE NAME INPUT DATE FOR $COPY
** LOAD VARIABLE PARM TO P37 WITH FILE NAME INPUT FOR $COPY
**
// EVALUATE P39-??48??
// EVALUATE P48-?48?-1
// EVALUATE P38,6-??48??
// EVALUATE P48-?48?-1
// EVALUATE P37-??48??
**
// IFF ?39?/     IF DATAF1-?39?                        EVALUATE CD-2030
// IFF ?38?/     IFF ?37?/     IFF DATAF1-'?37?,?38?'  EVALUATE CD-2030
// IF  ?38?/     IFF ?37?/     IFF DATAF1-?37?         EVALUATE CD-2030
// IFF ?50?/E                  GOTO NOEVMSG
// IF  ?CD?-2030 MSG ?USER?, FILE NAMED ?37? WAS NOT COPIED TO ?39? DURING EVOKED EXECUTION.
// TAG NOEVMSG
// IF  ?CD?-2030                        GOTO NOCOPY
**
// REGION SIZE-64
// LOAD $COPY
// IF ?38?>                             FILE NAME-COPYIN,LABEL-?37?,DATE-?38?
// ELSE                                 FILE NAME-COPYIN,LABEL-?37?
// FILE NAME-COPYO,LABEL-?39?
// RUN
// COPYFILE OUTPUT-SAME
// END
// TAG NOCOPY
**
// EVALUATE P48-?48?+6
// GOTO LOOPCP
// TAG ENDCPY
**
// IF ?L'502,2'?/?L'504,2'?             LOCAL OFFSET-502,DATA-'     '
// IF EVOKED-NO                         INFOMSG YES
// RETURN
***********************************************************************
**                                                                   **
**            MULTIPLE FILES COPY USING POP FILE UTILITY             **
**                                                                   **
** USES "POP" DISPLAY FILES UTILITY WITH NEW "S" CHARACTER AS OPCODE.  **
** FILES REQUESTED FOR "S" COPY WILL BE DISPLAYED ON PROMPT SCREEN WITH **
** THE OLD FILE NAME & DATE & NEW FILE NAME (DEFAULTS TO OLD NAME)     **
** NEW FILE NAME FIELD IS THE ONLY FIELD ALLOWED FOR INPUT BY USER     **
** MAXIMUM OF 12 FILES (ONE SET) MAY BE SET UP AT ONE TIME FOR "S" COPY. **
**                                                                   **
** RECOVERY NOTE: IF USER INTERRUPTS/CANCELS "S" RENAME REQUESTS, THEN **
**                LDA POSITIONS 502 THRU 506 SHOULD BE SET TO BLANKS TO **
**                CONTINUE TO ALLOW USE OF MULTI-COPY UTILITY.  THESE  **
**                POSITIONS ARE USED TO CONTROL ONE SET OF "S" COPIES. **
**                                                                   **
**      LDA NOTE: DO NOT RE-USE LDA POSITION 507 THRU 511 ALREADY USED **
**                FOR FILEQ CONTROL.  NEED TO KEEP POSITION 507 THRU 511 **
**                INTACT FOR LIFE OF FILE SET REQUEST WITHIN POP.      **
**                                                                   **
** PROCEDURES CALLED:                                                **
** -----------------                                                 **
** FILVPARM - INITIAL LOAD FILE/DATE INPUT & FILE OUTPUT TO VARIABLE PARM**
** FSQ      - JOBQ RUN OF THIS PROCESS IF REQUESTED DURING PROMPT    **
**                                                                   **
** PARAMETER DEFINITIONS.                                            **
** --------------------                                              **
```

```
** P01 - P36  FILE NAME AND DATE PARMS USED IN $COPY PROCESS.              **
** P01 - P12  REUSED AS NEW FILE NAME OUTPUT PARMS PASSED TO JOBQ CALL.    **
** P37 - WORKING PARM FOR $COPY FILE NAME INPUT.                           **
** P38 - WORKING PARM FOR $COPY FILE NAME INPUT DATE.                      **
** P39 - WORKING PARM FOR $COPY FILE NAME OUTPUT.                          **
** P40 - OP CODE POSITION IN LDA WHICH IS LOADED BY POP UTILITY.           **
** P41 - NTH PARM CONTAINING INPUT FILE NAME.                              **
** P42 - NTH PARM CONTAINING INPUT FILE DATE.                              **
** P43 - NTH PARM CONTAINING NEW OUTPUT FILE NAME.                         **
** P44 - COUNTER FOR TOTAL NUMBER OF CURR FILE REQUEST THAT ARE "S".       **
** P45 - NTH PASS FOR "S" COPY REQUEST WITHIN ONE FILE SET.                **
** P46 - FILE NAME POSITION IN LDA WHICH IS LOADED BY POP UTILITY.         **
** P47 - FILE DATE POSITION IN LDA WHICH IS LOADED BY POP UTILITY.         **
** P48 - WORKING PARM TO CONTROL $COPY VARIABLE PARM LOADING.              **
** P49 - WORKING PARM TO CONTROL $COPY LOOPING.                            **
** P50 - MEMORY FOR REQUEST FOR JOBQ OR EVOKED JOBS.                       **
** P51 - P63  ATTRIBUTE CONTROL PARMS USED FOR SCREEN PROMPT.              **
** P64 - PROCESS CONTROL CHARACTER PASSED TO LOAD VARIABLE PARM PROC.      **
**                                                                         **
** LDA USAGE:                                                              **
** ---------                                                               **
** 021 - 200  CURRENTLY USED BY FILE# PROC WITHIN POP AND ACCESSED HERE.   **
** 502 - 503  STORES TOTAL NUMBER OF "S" COPY REQUEST FOR ONE FILE SET.    **
** 504 - 505  NTH PASS WITHIN ONE FILE SET FOR "S" COPY REQUEST.           **
** 506 - 506  "S" COPY PROMPT & $COPY EXECUTION CONTROL.                   **
**            ALL "S" REQUESTS ARE IDENTIFIED AND SETUP FOR EXECUTION      **
**            DURING FIRST "S" REQUEST FOUND IN FILE SET.  ALL OTHER       **
**            ATTEMPTS TO RUN "S" REQUEST WITHIN ONE FILE SET WILL NOT     **
**            EXECUTE PROMPT & $COPY SINCE LDA 506 IS SET TO "S" AFTER     **
**            FIRST PASS THRU "S" REQUEST IN ONE FILE SET.  AFTER ALL      **
**            "S" REQUESTS HAVE PASSED LDA 502 THRU 506 IS BLANKED OUT.    **
*****************************************************************************
**
** END OF FILES PROCEDURE
```

Figure 14-38

Procedure FSQ

```
*****************************************************************************
**                            03-22-88  TIM A. HACK                        **
**                            EASTMAN KODAK/WWBIS                          **
**                            Proc Name: FSQ  (JOBQ REQUEST FROM FILES)    **
*****************************************************************************
**
** REPOSITION NEW OUTPUT FILE NAMES PASSED AS PARM 01 - 12 TO ORIGINAL PARMS
**
// EVALUATE P41,2-12                  INITIAL INCOMING PARM ID SET TO 12
// EVALUATE P43,2-00
**
// TAG REPOS
// EVALUATE P43,2-?41?*3              REPOSITION INCOMING PARM VALUE TO THIS PARM
// IFF ??41??/                        EVALUATE P?43?-??41??
// ELSE                               EVALUATE P?43?-''
// EVALUATE P41-?41?-1                COUNTDOWN NEXT INCOMING PARM TO REPOS
// IF ?41?>00                         GOTO REPOS
**
** INITIALIZE VALUES FOR VARIABLE PARM LOADING IN FILVPARM PROC.
**
// LOCAL OFFSET-502,DATA-'0000'       ZERO OUT LDA CTRL COUNTS AT 1ST PASS
// EVALUATE P44,2-00
// EVALUATE P40,3-021
// EVALUATE P41,2-00
// EVALUATE P64-'S'
**
** CALL PROC TO ASSIGN VALUES TO VARIABLE PARMS
**
// INCLUDE FILVPARM *ALL
**
// LOCAL OFFSET-502,DATA-'?44?'       SAVE HOW MANY FILES REQUESTS FOUND
**
```

```
** COPY ALL QUEUED FILES FOR COPY WITH $COPY CALL WITH VARIABLE PARM LOADING
**
// EVALUATE P48,2-04
**
// TAG LOOPCP
// EVALUATE CD-0000
// EVALUATE P48-?48?-1
// EVALUATE P49-?48?/3
// IF ?49?>?L'502,2'?                GOTO ENDCPY
**
** LOAD VARIABLE PARM TO P39 WITH FILE NAME OUTPUT FOR $COPY
** "                    " P38 WITH FILE NAME INPUT DATE FOR $COPY
** "                    " P37 WITH FILE NAME INPUT FOR $COPY
**
// EVALUATE P39-??48??
**
// EVALUATE P48-?48?-1
// EVALUATE P38,6-??48??
**
// EVALUATE P48-?48?-1
// EVALUATE P37-??48??
// IFF ?39?/      IF DATAF1-?39?                         EVALUATE CD-2030
// IFF ?38?/      IFF ?37?/      IFF DATAF1-'?37?,?38?'  EVALUATE CD-2030
// IF  ?38?/      IFF ?37?/      IFF DATAF1-?37?         EVALUATE CD-2030
// IF  ?CD?-2030 MSG ?USER?, FILE NAMED ?37? WAS NOT COPIED TO ?39? DURING JOBQ EXECUTION.
// IF  ?CD?-2030                     GOTO NOCOPY
**
** PERFORM $COPY WITH VARIABLE LOADED PARMS 37, 38 AND 39.
**
// REGION SIZE-64
// LOAD $COPY
// IF ?38?>                          FILE NAME-COPYIN,LABEL-?37?,DATE-?38?
// ELSE                              FILE NAME-COPYIN,LABEL-?37?
// FILE NAME-COPYO,LABEL-?39?
// RUN
// COPYFILE OUTPUT-SAME
// END
// TAG NOCOPY
**
// EVALUATE P48-?48?+6
// GOTO LOOPCP
// TAG ENDCPY
**
// RETURN
**************************************************************************
**                                                                      **
**     MULTIPLE FILES COPY USING POP FILE UTILITY JOBQ REQUEST          **
**                                                                      **
** THIS IS JOBQ PROC EXECUTED IF FILES USES JOBQ COMMAND KEY REQUEST.   **
** SEE FILES PROC FOR MORE COMPLETE DESCRIPTION.                        **
**                                                                      **
** PARAMETER DEFINITIONS:                                               **
** ----------------------                                               **
** SEE FILES PROC DOCUMENTATION FOR PARM DEFINITIONS.                   **
**                                                                      **
** LDA USAGE:                                                           **
** ---------                                                            **
** SEE FILES PROC DOCUMENTATION FOR LDA MAPPING FOR THIS ENTIRE FUNCTION.**
**************************************************************************
**
** END OF FSQ PROCEDURE
```

Figure 14-39
Procedure FILVPARM

```
**************************************************************************
**                         03-13-88  TIM A. HACK                        **
**                         EASTMAN KODAK/WWBIS                          **
**                         Proc Name: FILVPARM                          **
**************************************************************************
**
**
// TAG LOOP10
// IF ?L'?40?,1'?/?64?               EVALUATE P44,2-?44?+1
// ELSE                              GOTO SKIP10
// EVALUATE P63-?44?+50
```

```
// EVALUATE P?63?-''
** BEGINNING:
**   - FILE REQUEST IN LDA 21 WITH INCREMENT TO NEXT POP FILE REQUEST - 15
**   - FILE NAME IN LDA 22 IS 1 GREATER THAN ASSOCIATED FILE LDA REQUEST
**   - FILE DATE IN LDA 30 IS 9 GREATER THAN ASSOCIATED FILE LDA REQUEST
**
// EVALUATE P41-?41?+1
// EVALUATE P42,2-?41?+1           SETUP INPUT FILE DATE PARM ID
// EVALUATE P43,2-?41?+2           SETUP OUTPUT FILE NAME PARM ID
// EVALUATE P46,3-?40?+1           SETUP LDA START POSITION FOR FILE NAME
// EVALUATE P47,3-?40?+9           SETUP LDA START POSITION FOR FILE DATE
**
// EVALUATE P?41?-?L'?46?,8'?      ASSIGN INPUT FILE NAME TO NTH PARM-P41
// IFF ?L'?47?,6'?/                EVALUATE P?42?,6-?L'?47?,6'?
// ELSE                            EVALUATE P?42?-''
// IF ??43??/                      EVALUATE P?43?-?L'?46?,8'?
// EVALUATE P41-?41?+2
**
// TAG SKIP10
// EVALUATE P40,3-?40?+15
// IF 187>?40?                     GOTO LOOP10
// RETURN *ALL                     THANKS JWC FOR *ALL PARM ON RETURN
**************************************************************************
**                                                                      **
** SEE FILES/FILEQ PROC FOR MORE COMPLETE DESCRIPTION.                  **
**                                                                      **
** PARAMETER DEFINTIONS:                                                 **
** --------------------                                                 **
** SEE ABOVE PROC DOCUMENTATION FOR PARM DEFINITIONS.                   **
**                                                                      **
** LDA USAGE:                                                           **
** ---------                                                            **
** SEE ABOVE PROC DOCUMENTATION FOR LDA MAPPING FOR THIS FUNCTION.      **
**************************************************************************
**
** END OF FILVPARM PROCEDURE
```

Improving POP's File Delete

by Martin Bell

Code on diskette:
Procedure FILEZ

POP has a new S/36 operand — Z for ZAP — to delete a file from the FILE screen without having to go through the confirmation step required by operand D. Using Z is significantly faster than using D, especially when you have to delete several files.

To use Z, simply create procedure FILEZ (Figure 14-40). The POP tutorial explains how the FILE screen's header can be edited to include Z and its description.

Figure 14-40

Procedure FILEZ

```
* POP FILEZ: Zap a file with no confirmation
// LOAD $DELET
// RUN
// SCRATCH LABEL-?1?,
// IFF ?2?/ DATE-?2?,
// UNIT-F1
// END
```

Improving File and Library Save in POP

answered by Jeff Silden

Q When our shop uses the Library facility of IBM's POP to FROMLIBR library members onto diskette, we use a volume ID other than IBMIRD. Consequently, we must always rekey the shop's standard diskette volume ID in place of the IBM "standard" — and we are tired of it! Similarly, we use POP's File facility to save files onto diskettes, and every time we use the K option to save a file, we must key the diskette volume ID. Isn't there some way we can make our standard the default?

A Your frustrations with POP can be cleared up fairly easily. To make your shop's standard volume ID the default for the Library facility, change line 12 of procedure LIBRK in #POPLIB as shown in Figure 14-41. In the figure, I have used BACKUP as the standard volume ID; however, you can change IBMIRD to any valid six-character value.

For your file saves to diskette through POP's option K, you might want to add the parameter shown in Figure 14-42 to line 3 of procedure FILEK in #POPLIB. This change causes your six-character volume ID number to be displayed automatically each time you use the File facility to save a file onto diskette. As with the change to procedure LIBRK, you can use any six-character value in place of the word BACKUP.

Figure 14-41
Modifications to procedure LIBRK

```
// IF ?3?/O GOTO LIBRARY
// IF ?1?/ RETURN
*
// IFF ?C1?>4 GOTO NOPERIOD           CAN'T BE NAME.ALL IF NOT LONGER THAN 4
// LOCAL DATA-'?1?',OFFSET-301        PUT NAME AT 301 IN LDA
// EVALUATE P62-?C1?-4                LENGTH OF POSSIBLE PARTIAL NAME
// EVALUATE P63-301+?62?              STARTING POINT OF POSSIBLE '.ALL'
// IF ?L'?63?,,4'?-' ALL' EVALUATE P1-?L'301,?62?'? P63-ALL    IS '.ALL' THERE?
*                                     IF SO, P1-PARTIAL NAME AND P63-'ALL'
// TAG NOPERIOD
// IF ?2?/L EVALUATE P2-LIBRARY
** EVALUATE P7-?3? P3-?1? P4-I1 P5-999 P6-IBMIRD   <--- ORIGINAL CODE
// EVALUATE P7-?3? P3-?1? P4-I1 P5-999 P6-BACKUP   <--- NEW CODE
// IF ?3?-ALL  EVALUATE P3-''       FILE NAME ALL NOT ALLOWED
// IF ?6?/ALL HELP FROMLIBR ?1?,?63?,?2?,?3?,?4?,?5?,?6?,?7?,?8?,?9?,?10?,+
   ?11?,?12?                        PARTIAL NAME.ALL WAS SPECIFIED
// ELSE HELP FROMLIBR ?1?/,?2?,?3?,?4?,?5?,?6?,?7?,?8?,?9?,?10?,?11?,?12?,?13?
// IF ?CD?-1991 RETURN
// IF ?CD?-2143 RETURN
LIBRPARM ,,?L'201,100'?
// EVALUATE P5-ADD  P50-'*'         COMMENT OUT ALL FROM LIBR'S
// TAG 1
```

Figure 14-42
Modifications to procedure FILEK

```
// LIBRARY NAME-O
// MEMBER USER1-##MSG2
HELP SAVE ?1?,,?2?,BACKUP                    <<-- MODIFIED CODE
```

Restricting POP's File Display with a File Mask

by Carl W. Selley

Code on diskette:
Procedure FILEB

POP procedure FILEB gives you a quick POP file group display restricted to a specified group. Once you've placed procedure (Figure 14-43) in #LIBRARY, you can use this tool by entering FILEB and the name of the file group you want displayed; the file group name can be in any of the following formats:

```
FILEB ABC
FILEB ABC.
FILEB .ABC
FILEB .ABC.
```

This brings up the POP display limited to the group you specified. To redefine the group, use POP's Command key 5 option.

Figure 14-43
Procedure FILEB

```
* FILEB - Restrict POP display to the group specified in P1
// LOCAL OFFSET-1,BLANK-8,DATA-'?1?'
// LOCAL OFFSET-9,DATA-' I
// LIBRARY NAME-#POPLIB
// RESET FILE#
```

Browsing Spool Files with POP

by Bret B. Myrick, Sr.

Code on diskette:
Procedure DSP

Using the COPYPRT procedure to look at a spool ID can be tiresome because of COPYPRT's limited paging, character search, and positioning techniques. If you have POP, however, I can help you with a better method for viewing spool entries.

Procedure DSP (Figure 14-44) uses the $UASF utility just like COPYPRT, but the similarity ends there. After copying the spool entry to a disk file, the procedure calls the File procedure in #POPLIB. Now you have all the Roll key, paging, and character search operations you have when you browse a disk file with POP.

When it's time to end procedure DSP, type XZ at the top of the browse screen to execute procedure FILEZ (see *Improving POP's File Delete,* page 446.)

Figure 14-44

Procedure DSP in #LIBRARY

```
**
*---------------------------------------------------------------------*
*  Turn off the INFOMSG and ensure that there is a SPOOL ID or CANCEL *
*---------------------------------------------------------------------*
// INFOMSG NO
// IF ?1R?= CANCEL
*----------------------------------------------------------------------*
* Delete any SPOOL ID that may already exist for this workstation      *
*----------------------------------------------------------------------*
// IF DATAF1-SPOOL.?WS? DELETE SPOOL.?WS?,F1
*-------------------------------------------------------*
*  Load $UASF and convert the SPOOL ENTRY to a disk file *
*-------------------------------------------------------*
// LOAD $UASF
// RUN
// SPOOL SPOOLID-?1?,NAME-SPOOL.?WS?
// END
*---------------------------------------*
*  Run the FILE procedure in #POPLIB    *
*---------------------------------------*
// LIBRARY NAME-#POPLIB
FILE SPOOL.?WS?
**
```

Improving and Adding Operations in POP

by Matthew Henry

Code on diskette:

Procedures FILEE, FILEL, FILEQ2, FILEU, FILEKY6, LIBRA, LIRCOMP, LIBRKY8

I have added several opcodes and command keys to POP to help me be more productive while accomplishing some of the small tasks I must perform daily. Several of the following POP procedure commands help me do up to 12 things at once.

• FILEE — changes the current E operation code from DFU edits to Query Data Entry Facility (QRYDE) edits. To use FILEE, you must configure Query/36 on your system, and the file to edit must be linked to an IDDU definition. Using QRYDE instead of DFU for file edits saves library space because you do not have to keep any DFU library members.

```
// QRYDE ?1?,?2?
// RETURN
* CALLS QUERY/36 DATA ENTRY FACILITY
* MUST HAVE QUERY/36 INSTALLED ON THE SYSTEM
```

Note: Since #POPLIB already contains a FILEE (call DFU) procedure, you should rename either it or the FILEE (call Query) procedure before installing this procedure in #POPLIB.

• FILEL — links files to IDDU definitions. You must configure IDDU on your system and have a file definition for the file to be linked. Accessing IDDULINK from POP lets you perform up to 12 links at one time and does not require the file name to be rekeyed.

```
// HELP IDDULINK,LINK,?1?,MASTER,,?2?
// RETURN
* POP CODE FOR LINKING TO IDDU DEFINITION
```

• FILEQ — displays a file through Query/36 with IDDU field headers. You must configure Query/36 on your system and the file to view must be linked to an IDDU file definition. Query/36 file displays show the predefined headings for all the fields and display binary and packed fields in the IDDU formatted form (e.g., slashes for dates, colons for time, and commas in the right place).

```
// QRYRUN ,,?1?,DISPLAY
// RETURN
* DISPLAYS A FILE WITH IDDU HEADINGS
* MUST HAVE QUERY/36 ON THE SYSTEM
```

Note: This procedure is named FILEQ2 on diskette. To use it in #POPLIB, you must rename it to FILEQ. Another procedure described in Renaming and Copying Multiple Files in POP, *page 434, uses the name FILEQ, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.*

• FILEU — unlinks files from their IDDU file definition. You must configure IDDU on your system, and the files must be linked to an IDDU file definition. You can accomplish 12 unlinks at a time by unlinking files with POP. FILEU complements the operation of the E, L, and Q codes.

```
// IDDULINK UNLINK, ?1?
// RETURN
```

• FILEKY6 — switches to POP's library facility from the file facility using Command key 6. Command key 6 saves you time by letting you access the POP library display directly without having to return to a menu.

```
POPPR,#LIBRARY ,LIBR,3
// RETURN
* PROMPTS FOR LIBRARY FOR POP LISTING
```

• LIBRA — reallocates a library using SSP's ALOCLIBR procedure. The A code can be used only from POP's full library display (option 1 on the POP menu). By using the A code, you can reallocate up to 12 libraries at a time. It, along with the following C code, complements the library management functions of POP.

```
// IFF ?3?/O RETURN
// HELP ALOCLIBR ?1?
// RETURN
* POP CODE FOR ALLOCATING LIBRARY
```

• LIBRCOMP — condenses a library from POP's LIBR display using SSP's CONDENSE procedure. Add these statements to the beginning of the existing POP LIBRC procedure. With LIBRC, you can condense up to 12 libraries at a time without rekeying the name each time.

```
// IFF ?3?/0 GOTO COMPILE
// * 'Condensing ?1? library.'
// CONDENSE ?1?
// RETURN
// TAG COMPILE
```

• LIBRKY8 — switches to POP's file facility from the library facility using Command key 8. Command key 8 along with Command key 6 (i.e., FILEKY6) make it easy to switch back and forth between POP's file and library facilities without having to exit to a menu.

```
FILE,#POPLIB
```

or

```
// INCLUDE FILE,#POPLIB
```

You must save these procedures whenever you install a new release of POP or apply a PTF that patches one of the changed procedures. And it is a good idea to make a backup of the entire #POPLIB library.

Printers

CHAPTER

15

Opening and Closing Printer Files in RPG

answered by Mike Patton and Gary T. Kratzer

Code on diskette:

Assembler subroutines SUBROP, SUBRCL

Q While using an order entry program, I want to print a form without ending the program and to share the same printer among multiple terminals. When I use the DEFER-NO option on the // PRINTER statement in my OCL, one terminal locks up the printer. I don't want to convert my order entry program to a MRT because I use substitution parameters in my OCL. Is there a method to close the print file and reopen it within my RPG II program or some other method that lets multiple terminals on the S/36 share the same printer?

A One solution is using an assembler subroutine to open and close the print file. The following code shows the calling sequence for subroutines SUBROP and SUBRCL that let an RPG program open and close a file at will:

```
C                    EXIT SUBROP               (or SUBRCL)
C                    RLABL          FLNAME  8  (contains the file name)
```

Make sure you reopen the printer file before output is attempted or the job ends — or the RPG program will generate an error.

Re-creating Subroutine SUBROP

If you don't have assembler subroutine SUBROP, you can re-create it with procedure MKSUBROP (you don't need IBM's Assembler Language Program Product to install SUBROP). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBROP. You need to run MKSUBROP only once to create the SUBROP subroutine.

```
// * 'Re-creating R-module SUBROP in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'     Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9D6D74040000000400000000000000102000200990002300000000000'
// LOCAL OFFSET-273,DATA-+
'00000000000000000803F980000000000000000000C3C2000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBROP member to insert object code
// LOAD $FEFIX
// RUN
```

Continued

```
HDR   3892 SUBR000000
PTF   A888 RSUBRDP,99. #RPGLIB
DATA 0862 00 0000 E2D8E7E4C2D9D8070000000600000000000000000000000000000000000000
DATA 9112 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA E132 00 0040 E3340034340100403402006134080065750299850206350100667501023007 18
DATA 0328 00 0080 00F2810080FF08F2811185020700B70019200100680780C010F00302C1608070 3
DATA 64E3 00 0080 E3200068006 9F401340CF401040280010700580C010088006702010000C20200
DATA 9CF4 00 0040 000087000000003FFFF000000000000000000000000000000000000000141206 01
DATA 6B7E 00 00C0 C5FFF16C17773E7C40FF6C88FEFF751018F00000F0000003000008B300000620C
DATA 7849 00 00E0 0000840082C0870060000000000000000000000000000000000000000000000000
DATA 880A 00 0100 0000006C17773E7C40FF6C88FEFF7510 18F00000F00000300000680000008 20C
DATA 128A 00 0120 0000840082C0000060000000000000000000000000000000000000000000000000
DATA 0980 00 0140 000000000000600002CC610000830000000000000000000000000000003F330000
DATA E0D1 00 0160 000000000000C097C07C7F2F340404000007F30000000000000C20000020061000 3
DATA 1986 00 0180 3000000000000000000000000000000803F800000000000000000D97CD7C7F4F040
DATA 6F43 00 01A0 4C400007F80000000000000000200000200610003300000000000000000000000 0
DATA E8BC 00 01C0 00803F9800000000000000000097CD7C7F4F140404000007F9000000000000000200
DATA FC17 00 01E0 00020061000830000000000000000000000000000000803FC0000000000000000FF
END   A1#6
```

Re-creating Subroutine SUBRCL

If you don't have assembler subroutine SUBRCL, you can re-create it with procedure MKSUBRCL (you don't need IBM's Assembler Language Program Product to install SUBRCL). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRCL. You need to run MK SUBRCL only once to create the SUBRCL subroutine.

```
// * 'Re-creating R-module SUBRCL in library #RPGLIB '
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'0000007E'      Number of #MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9C3D340400000040000000000000102000200890007300000000000'
// LOCAL OFFSET-273,DATA-+
'0000000000000000803F9800000000000000000C3C2000000'
// LOAD MAKMEM
// FILE NAME-BINARY LABEL-#MAINT,RETAIN-J,BLOCKS-26 EXTEND-26
// RUN
* Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRCL member to insert object code
// LOAD $FEFIX
// RUN
HDR   3848 SUBRC00000
PTF   132A RSUBRCL,99 ,#RPGLIB
DATA 5038 00 000C E2D8E2E4C2D9C3D3000000000580000000000000000000000000000000000000
DATA DA0C 00 002C 0000000000000000000000000000000000000000000000000000000000000000
DATA 212F 00 004C E33400343401004834020040340800617502988502063501006175010279007 1C
DATA 4378 00 006C 00F28100B0FF0BF2811A8802D9C0B7001920010067098C0109000302C15080703
DATA B548 00 008C E3220057005 5F401040380010900570E010051006 3C2010000C202000C08700
DATA 1F91 00 00AC 00000 3FFFF0000000000000000000000000000000000000000000000010EDA01
DATA B3CC 00 00CC C6FFFF57000000000000000000000000000000000000000000000700000000000
DATA A302 00 00EC 0000000000000000000000000000000000000000000000000000000000000000
DATA 0386 00 010C 000000 5C17773E7C40FF6C88FEFF7510 18F00000F0000C30000088000000820C
DATA 4177 00 012C 0000640082000000800000000000000000000000000000000000000000000000
DATA 0641 00 014C 000000000000500000200051000830000000000000000000000000000803F330000
DATA 0E4E 00 0160 000000000000C097CD7C7F3F340404000007F30000000000000700000200610003
```

Continued

```
DATA 4834 00 0180 [illegible]
DATA 89DC 00 01A0 40400007F6000000000000C2000000200C81000330000000000000000000000000
DATA 1B50 00 01C0 00803F9900000000000000000007C07C7F4F14040400007F90000000000000200
DATA 2AAB 00 01E0 [illegible]
END  0802
```

Retrieving the Spool ID

by Mel Beckman

Code on diskette:
Procedure TESTSX
RPG program TESTSX
Assembler subroutine SUBRSX

Often it is useful for a program to know the spool ID for a printer file it creates. For example, suppose that after a certain report program runs, you want the spooled printout copied automatically to a disk file for later perusal at a workstation. A common method would be to specify a unique forms number on the // PRINTER statement, and then use the Fxxxx option of COPYPRT to copy the spool file for that specific forms number. However, if you later decide to print the spool entry, you will have to deal with a forms change message. Worse, if other spool entries have the same forms number (perhaps from a previous run), those spool entries also will be copied.

Clearly, a better method would be to obtain the spool ID for the desired print file while the program is running. The program then could pass the spool ID (SPxxxx) to a procedure via the LDA, and a COPYPRT could be issued for that specific spool ID.

It turns out that the spool ID value is available to any program via the $INFO supervisor call. A simple assembler language subroutine can make $INFO retrieve the spool ID and pass the result back to the RPG program.

To use the new subroutine, code the following three statements into your RPG program where you want to retrieve a spool ID:

```
C                      MOVE 'REPORT'  SPOOL#  8
C                      EXIT SUBRSX
C                      RLABL          SPOOL#
```

The first statement moves the name of the printer file (from positions 7 through 14 of the F-specs) into the eight-character, dual-purpose field SPOOL#. The second and third statements exit to subroutine SUBRSX, which returns the spool ID to the leftmost six characters of SPOOL#. When the program ends, it should store the SPOOL# field in the LDA so that the calling procedure can reference it via an LDA substitution expression.

If you need the spool ID for several printer files, include in your program everywhere you need them the three statements shown above. Because you specify the printer file name in the first statement, spool IDs can be returned for any of several printer files that the program might contain.

Figures 15-1a and 15-1b show a complete sample program and procedure illustrating the technique. Note that the // PRINTER statement specifies PRIORITY-0 to hold the printout on the spool queue. If PRIORITY-0 is not used and the spool entry starts printing, the COPYPRT procedure will fail.

Complete documentation for $INFO is contained in the *S/36 Programming with Assembler* manual.

Figure 15-1a

Sample program TESTSX

```
.        1         2         3         4    .  5         6         7         8
     H    014                                                                TESTSX
     F*
     F* Program to demonstrate the use of SUBRSX
     F*
     FREPORT  O          132            PRINTER
     I*
     I* When the program ends, the LDA will contain the spool-ID
     I*
     I            UDS
     I                                          1    8 SPOOL#
     C*
     C* Retrieve the spool-ID for our printer file, named 'REPORT'
     C*
     C                     MOVE 'REPORT  'SPOOL#  8
     C                     EXIT SUBRSX
     C                     RLABL
     C*
     C                     SETON                     LR
     O*
     O* These output specs simply print a line on the report
     O*
     OREPORT  T        LR
     O                                     11 'SPOOL-ID IS'
     O                          SPOOL#     20
```

Figure 15-1b

Sample procedure TESTSX

```
**
** This procedure demonstrates the use of SUBRSX
**
// LOAD TESTX
// PRINTER NAME-REPORT,PRIORITY-0
// RUN
**
** The spool entry produced by TESTX is now held on the print queue,
** and the LDA contains the spool-ID in positions 1-6  Use an LDA
** substitution parameter to make COPYPRT copy the specific entry we want
**
// IF DATAF1-SPDATA DELETE SPDATA,F1
COPYPRT ?L'1,6'?,SPDATA
```

Re-creating Subroutine SUBRSX

If you don't have assembler subroutine SUBRSX, you can re-create it with procedure MKSUBRSX (you don't need IBM's Assembler Language Program Product to install SUBRSX). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRSX. You need to run MKSUBRSX only once to create the SUBRSX subroutine.

Continued

```
// * 'Re-creating R-module SUBRSX in library #RPGLIB
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000039'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9E2E7404000000020000000000000001020002009900013000000000000
// LOCAL OFFSET-273,DATA-+
'00000000000000000803FCC0000000000000000C3C2000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRSX member to insert object code
// LOAD $FEFIX
// RUN
HDR  38C8 SUBRS00000
PTF  0CE8 RSUBRSX,99,,#RPGLIB
DATA AACF 00 0000 E20BE2E4C2D9E2E7000000006800000000000000000000000000000000000000
DATA 3395 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA FC18 00 0040 E3300030340800343401002C3402003035010034 1C010038020F010038003FC2
DATA 7A08 D0 0060 020035F401040F0E0100340030C2010000C20200000028261E1A18130F080703
DATA 5080 00 0080 E30E003FC08700000013000000000000030007000052895006F21003BCF2523C
DATA 4199 00 00A0 000000AE016408C2010000C2020000C0B7000000000000000000000000000000
DATA 416B 00 00C0 C500005C17773E7C40FF5C86FEFF75101BF00000F000003000006B000000620C
DATA 6479 00 00E0 0000640062C0000060000000000000000000000000000000000000000000000
END  16D0
```

Resetting Page Numbers

answered by Ron Mendel

Q We print a report that summarizes the weekly sales activity for each of our sales representatives. As the report is printed now, the pages are numbered consecutively from the beginning of the report. We would like to begin renumbering the pages each time the sales representative changes. Is there a way to do this in RPG II?

A In RPG II, you use the special pagination word PAGE in positions 32 through 37 of the O-specs to cause automatic numbering of pages. If you condition PAGE on an output indicator, when the indicator is on, the PAGE field is reset to zero, and 1 is added to the field before it is printed.

Figure 15-2 shows an example of conditioning PAGE on an L3 control break. When L3 is on, pagination begins again at 1. If there is overflow and L3 is not on, the PAGE field is not reset to zero; instead, 1 is added to it before it is printed.

For further information, see the Special Words section of the Output Specifications chapter in the S/36 *RPG II Reference Manual*.

Figure 15-2

Sample O-specs to begin renumbering pages

```
*... . 1 . . 2 . 3 4 . . 5 . .. 6 .. 7 . . 8
     O         D  201   L3
     O        OR  201   OV
     O                         SMNAME     30
     O                                    75 'PAGE'
     O                  L3     PAGE       80
```

Numbering Pages

by Richard Comstock

When using the RPG page-number fields PAGE through PAGE7 to number pages on a report, it helps to understand how the fields function with a conditioning indicator. If the conditioning indicator is off, 1 is added to the page number value before it is printed. (Turning off the conditioning indicator does not prevent printing of the values of the page-number fields.) If the indicator is on, the page number is reset to 1.

This S/36 technique is useful when you need to reset page numbers (e.g., each salesman's or division's data is to begin with page 1).

Forcing Printer Overflow

by Paul Sherrill

The RPG program cycle simplifies production of standard reports, but sometimes you need to take control from the cycle and put it in the hands of the programmer. For example, you might want to control the printing of heading lines by using the EXCPT operation.

The partial program shown in Figure 15-3 causes exception time output (header) by executing an EXCPT operation. The report heading is printed when the overflow indicator is set on and "fetch overflow" is specified with an F in column 16. The desired spacing is specified in the header specification; the 0s in columns 17 and 18 prevent any additional line skips. Because the overflow indicator is set on in the calculation specifications, the exception line causes the header to print at the top of each new page.

Figure 15-3

Forcing printer overflow with an EXCPT

```
*     . 1 .. ... 2       3         4         5         6         7 .       8
     C*
     C*
     C*
     C                         SETON                     OF
     C                         EXCPTHEADER
     C                         SETOF                     OF
     C*
     C*
     C*
     OREPORT  H 2301   OF
     O                          NAME       15
     O                          ADDR1      50
     O          EF00            HEADER
     O*
     O*
     O*
```

Printing Boldface

answered by Ron Mendel

Q Is there some way we can print boldface on a printer that does not have a special double strike or boldface code? We have a vendor report in which we want to boldface the line containing the vendor name so it stands out from the rest of the information.

A In a report produced by an RPG program, to print an output line boldface, you need to use two almost identical O-specs for the same line of output. In the first line, include a zero in position 18 (space after print) to tell the printer to space zero lines after printing. In the second O-specs, use 1, 2, or 3 in position 18, depending on how far you want the printer to space after the boldface line. Figure 15-4 shows O-specs that print VNAME boldface and then space two lines before printing anything else.

Figure 15-4
Sample O-specs to print boldface

```
* . . . 1 . . . . 2 . . . . 3 . . . . 4 . . . . 5 . . . . 6 . . . . 7 . . . . 8
     O          D  0        L3
     O                             VNAME      30
     O          D  2        L3
     O                             VNAME      30
```

Printing Report Lines Using Arrays

by James H. Hamby

I have a method of using a compile time array to reduce the number of coding lines needed in O-specs. Normally, you would have to code the headings for a report as shown in Figure 15-5. Each column heading must be defined in a separate O-spec. If you use extensive headings, this practice can mean many O-specs.

However, if you use a compile time array (Figure 15-6), you can print the entire heading with only one line of code by referring to the heading the same way you would to a group of numbers in an array. You need an extension specification to describe your array, and at the end of your program — in the array — you need your heading line.

If you use compile time arrays to print headings, adjustments or corrections can be made quickly and easily. Once you have your array typed in, you can print any line of the array as many times as needed by referring to its line number. Think how easy it would be to print whole lines of underlining or asterisks to spruce up your reports.

The maximum length of a compile time array that uses alphanumeric information is 96 characters. Thus, if you need to use the full 132 characters for your report headings, you simply use two arrays, halving the heading information.

If you don't want to compile your program again after changing your array, you might consider using a pre-execution time array.

Figure 15-5

Coding O-specs for a heading

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... .. 6 ... ... 7 ... ... 8
     OPRINT   H  306   OANL2
     O       OR        L2
     O                                  8 'DATE'
     O                                 18 'ACCOUNT'
     O                                 28 'N A M E'
     O                                 46 'BALANCE'
```

Figure 15-6

Coding an array for a heading

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
     E                          J  1   1 50
     .
     .
     .
     OPRINT   H  306   OANL2
     O       OR        L2          J,1     50
     O
     .
     .
     .
**
       DATE     ACCOUNT   N A M E           BALANCE
```

Printing Lines and Dashes

by Deborah A. Kacerek

To print a line of dashes (or asterisks or double lines) across 132 columns, I use the method illustrated in Figure 15-7. In the E-specs, I define an array of 132 elements. In the C-specs, during the first pass, I move a dash, or whatever character is required, into this array. In the O-specs, all I need to reference is array DASH to print dashes across the 132 columns.

I also use this method to print single and double lines for column totals as illustrated in Figure 15-8. This method is especially helpful because if the program requires single lines at one level (e.g., section totals) and double lines at another (e.g., grand total), all you have to do is move either a dash or an equal sign into the array at the appropriate level break.

Figure 15-7

Print a line of 132 dashes

```
*... ... 1 ... .. 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
     E*  ARRAY DASH - PRINTS LINE OF 132 DASHES
     E                  DASH       132  1
     E**
     C**
     C** FIRST PASS OPERATIONS
     C         81          SETOF                         80    1ST PASS
     COMPLETE
     C        N81          SETON                         8081  FIRST PASS
     C*
     C         80          MOVE '-'       DASH
     C**
     O** HEADING OUTPUT
     O**
     OPRINT  H 11      80
     O      OR         L1       DASH     132
```

Figure 15-8

Print single underscore or double underscore at control breaks

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... .. 6 ... ... 7 ...   .. 8
     E*  ARRAY ULINE - PRINTS COLUMN TOTAL UNDERLINES
     E                  ULINE        9  1
     C***
```

```
C         80           MOVE '-'       ULINE
C**
CL1                    MOVE '-'       ULINE
C**
O** PRINT COLUMN UNDERLINES
OPRINT  T  1      L1
O                          ULINE     51
O                          ULINE     61
O                          ULINE     71
O                          ULINE     81
O                          ULINE     91
O                          ULINE    101
O                          ULINE    111
O                          ULINE    121
O                          ULINE    131
```

Printing a Sample Report from O-Specs

by Perry Gardai

program by Ernie Malaga

Code on diskette:

Procedures @RPTSMPL, REPTSMPL
RPG programs SMPLA@, SHRTAR
Screen format member REPTSMPL

There are times when you need a sample printout of, say, your gross payroll report — or a sample of a report generated by any S/36 program that contains a file specification that defines a printer file. One or two pages from a live run of the program would suffice, but running a live program might expose sensitive data or update the files while producing a printout or just take too long to print. Wouldn't it be great to have a utility that produces a mockup of the printout similar to the way SDA's option 8 prints a sample display? REPTSMPL is such a utility.

Screen format member REPTSMPL, procedures REPTSMPL and @RPTSMPL, and program SMPLA@ constitute utility REPTSMPL. Briefly, procedure @RPTSMPL creates a data file to hold the generated source code of any RPG II program that contains up to eight printer files. Next, program SMPLA@ creates program SMPLB@ from the generated RPG source code by using the entire output specifications as the output for program SMPLB@. Finally, program SMPLB@ produces the sample reports. That's it. The few intermediate steps involved in getting from the original RPG program to the report are detailed below.

Getting Started

Type in REPTSMPL to bring up the prompt screen (Figure 15-9; see Figure 15-10 for the screen format member) and start master procedure REPTSMPL (Figure 15-11). Enter the name of your source program and the name of the library in which the source program is stored, and then set up to six internal indicators that may influence conditional printing. Conditional printing refers to an O-specs field conditioned by specific indicator settings.

Figure 15-9

Prompt screen for procedure REPTSMPL

```
REPTSMPL PROCEDURE   PRINT A REPORT SAMPLE


ENTER THE FOLLOWING INFORMATION

1  NAME OF PRINTING PROGRAM            000000

2  NAME OF LIBRARY CONTAINING PROGRAM  00000000

3  INDICATORS TO BE SETON·             00   00   00   00   00   00
```

These settings must be on for the associated fields to print when the original program is executed as well as when the sample is produced by utility REPTSMPL. In our example program SHRTAR (Figure 15-12), indicators 12, 14, and 50 condition output fields and must be specified in the prompt screen if the related data fields are to be printed in the sample report.

The next several lines of procedure REPTSMPL edit and validate data entered into the prompt screen. The procedure issues appropriate error messages and redisplays the prompt screen if it detects any errors. Error messages indicate, for example, whether a library name is missing or the library is not in the VTOC. When the data passes muster, procedure REPTSMPL loads the LDA with the prompt screen values and submits procedure @RPTSMPL (Figure 15-13) to the job queue. It is in procedure @RPTSMPL that the real work begins.

What Happens in Procedure @RPTSMPL

The job flow of procedure @RPTSMPL begins with the first $MAINT routine, which creates data file SMPL1?WS? on disk. Program SMPLA@ (Figure 15-14) reads SMPL1?WS? to identify specific data elements such as the printer file name, array names, input field names, fields defined in the C-specs, and all the printer file O-specs. As program SMPLA@ processes each specification type, it branches to the appropriate subroutine to manipulate the data and then outputs to file SMPL2?WS? specific information relating to the data fields and O-specs.

Data file SMPL2?WS? contains RPG program SMPLB@ (Figure 15-15). Using the second $MAINT utility, procedure @RPTSMPL moves the data file version of program SMPLB@ from file SMPL2?WS? into a temporary library called WORKLIBR — which you can create if it doesn't exist — to avoid disturbing any active user libraries. After that, the RPGC utility compiles program SMPLB@. If the compile fails, the appropriate message is displayed, and source program SMPLB@ is listed and removed from WORKLIBR. If SMPLB@ compiles successfully, however, it is executed, and a sample version of the original report is printed (Figure 15-16). Finally, the procedure ends after removing both the object and source members of program SMPLB@ from WORKLIBR.

The F-spec from program SHRTAR is the printer output file for program SMPLB@. Likewise, all of the data fields identified from the E-, I-, and C-specs of program SHRTAR are defined in the C-specs of program SMPLB@. Program SMPLB@ uses arrays ALLX and ALL9 to move Xs into alphameric fields and 9s into numeric fields. Therefore, when the output section of program SMPLB@ is executed, the original data fields used in program SHRTAR are filled with the correct data representation characters, namely Xs and 9s. Then, all numeric fields defined in the C-specs of program SMPLB@ are Z-SUBed into themselves to ensure that negative signs will be printed.

Except for a few minor exceptions, the entire printer output section from program SHRTAR is copied into program SMPLB@. The exceptions include output lines defined in the original program as header, detail, or total lines; these lines change to exception lines without an exception name. Exception names also are dropped for exception output that was specified in the original program. In addition, fields associated with field-conditioning indicators in the output of the original program are included in the output of program SMPLB@ only if the associated field-conditioning indicator was specified on the prompt screen.

Limitations of Utility REPTSMPL

Before using utility REPTSMPL, you should be aware of several special considerations. First, utility REPTSMPL works only with printouts produced by RPG programs. Therefore, you cannot use it to create sample printouts of printed output produced by DFU or Query/36. Second, utility REPTSMPL cannot work with object members; the original source member of the program must be resident on the disk. Third, if the original report-producing program contains a WORKSTN file, REPTSMPL may produce unpredictable results because of the presence of alphameric names such as *STATUS in the *from* and *to* columns of the (INFDS) I-specifications. Also, the reserved names (e.g., UDATE, UYEAR, UMONTH, UDAY, PAGE, and PAGE 1 through PAGE 7) are used as is if found in the O-specs of the original program. Thus, the actual values rather than the appropriate character representations are printed in the sample report (e.g., UDATE will print as the actual session date rather than 99/99/99). In addition, entire arrays printed without an index reference in the original program are treated as a single field equal to the length of one element of the original array. Therefore, when printed in the sample report, the field (single array element) is positioned properly as the last element of the original array. And finally, utility REPTSMPL does not verify that the original program is coded properly. For example, if a binary field is assigned an invalid length (such as eight bytes), REPTSMPL won't question it. As a result, the 8-byte "binary" field is treated as an unpacked 8-byte numeric field.

With utility REPTSMPL, you can quickly generate a sample of any printout produced by an RPG program. With just a few special considerations to keep in mind, utility REPTSMPL can become a valuable addition to your data processing tool box.

Figure 15-10

Screen format member REPTSMPL

```
*       1         2         3         4         5         6         7         8
  SENTER          Y    YY  N      Y                          G
  0           43 1 2Y                                   CREPTSMPL PROCEOURE    PRX
  OINT A REPORT SAMPLE
  0           32 5 2Y                                   CENTER THE FOLLOWING INFX
  OORMATION
  D           28 7 2Y                                   C1  NAME OF PRINTING PROX
  DGRAM
  DPRGM        6 741Y  Y            Y         Y
  D           38 9 2Y                                   C2  NAME OF LIBRARY CONTX
  DAINING PROGRAM
  DLIBR        8 941Y  Y            Y         Y
  D           2611 2Y                                   C3  INDICATORS TO 8E SETX
  DON
  DIND         21141Y  Y            Y         Y
  DIND         21146Y  Y            Y         Y
  DIND         21151Y  Y            Y         Y
  OIND         21156Y  Y            Y         Y
  DIND         21161Y  Y            Y         Y
  DIND         21166Y  Y            Y         Y
  D           4823 2Y                                   CPRESS ENTER TO SUBMIT TX
  OO JOBQ OR CMO7 TO CANCEL
  DMSG        7524 2Y  Y          Y
```

Figure 15-11

Procedure REPTSMPL

```
** PROC-REPTSMPL   PRINT A REPORT SAMPLE
** Parameter and LDA usage
**
** Parm 1 - LDA 001-006   Name of original RPG-II program to be sampled
** Parm 2 - LDA 007-014 - Name of library containing the program
** Parm 3 - LDA 015-016 - First RPG indicator to be SETON
** Parm 4 - LDA 017-018 - Second RPG indicator to be SETON
** Parm 5 - LDA 019-020 - Third RPG indicator to be SETON
** Parm 6 - LDA 021-022 - Fourth RPG indicator to be SETON
** Parm 7 - LDA 023-024 - Fifth RPG indicator to be SETON
** Parm 8 - LDA 025-026 - Sixth RPG indicator to be SETON
** Parm 9 - Reserved for error messages
**
// TAG AGAIN
//   PROMPT MEMBER-REPTSMPL,FORMAT-ENTER LENGTH-'6,,2,2,2,2,2,2,75'
//   IF ?CD?-2007 RETURN
**
** Check input accuracy
//   IFF '?1?'- GOTO P1A
//   EVALUATE P9-'NAME OF PRINTING PROGRAM IS MISSING '
//   GOTO AGAIN
// TAG P1A
//   IFF '?2?'- GOTO P2A
//   EVALUATE P9-'NAME OF LIBRARY IS MISSING '
//   GOTO AGAIN
// TAG P2A
//   IF DATAF1-?2? GOTO P2B
//   EVALUATE P9-'LIBRARY NOT FOUND IN THE VTOC '
//   GOTO AGAIN
// TAG P2B
//   IF SOURCE-'?1?,?2?' GOTO P1P2
//   EVALUATE P9-'SOURCE PROGRAM NOT FOUND IN SPECIFIED LIBRARY '
//   GOTO AGAIN
// TAG P1P2
//   LOCAL BLANK-30,DATA-'?1?'
**
** All parameters pass the tests   Place their values in the LDA
//   LOCAL OFFSET-7,DATA-'?2?'
//   LOCAL OFFSET-15,DATA-'?3?'
//   LOCAL OFFSET-17,DATA-'?4?'
//   LOCAL OFFSET-19,DATA-'?5?'
//   LOCAL OFFSET-21,DATA-'?6?'
```

```
//   LOCAL OFFSET-23,DATA-'?7?'
//   LOCAL OFFSET-25,DATA-'?8?'
**
** Submit @RPTSMPL to the Job Queue.
//   JOBQ ,@RPTSMPL
```

Figure 15-12

Program SHRTAR

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ..  ... 8
     H                                                                          SHRTAR
     FSORTED  IR  F   3   3  3IT       EDISK
     FISHORT  IP  F 200 200R  I         DISK
     FCONTROL IC  F 200 200R31AI      1 DISK
     FSHRTAR  O   F 132 132      OA    LPRINTER
     FIDATEA  O   F  40  40             DISK                        A
     FIDATEB  IC  F  40  40R            DISK
     E    SORTED  ISHORT
     LSHRTAR    66FL 570L
     IISHORT  NS  01
     I                                        2  16 PART# L1
     I                                       17  17 WHSE# L1
     I                                      110 112 PLANN L1
     I                                       18  230DUEDT
     I                                       24  24 PROD
     I                                       25  25 ORTYP
     I                                       25  31 ORDER
     I                                       32  61 DESCR
     I                                       62  76 PARNT#
     I                                       77 106 PDESC
     I                                      107 107 TYPE
     I                                      108 109 CLASS
     I                                    P 110 1120PLANR
     I                                      113 137 CNAME
     I                                      138 138 PRTY
     I                                      139 140 UMEAS
     I                                      141 142 STAT
     I                                      143 152 REF#
     I                                      153 158 CJOB#
     I                                    P 159 1620QBO
     I                                    P 163 1660QOH
     I                                    P 167 1700QOOM
     I                                    P 171 1740QOOP
     I                                    P 175 1780ALLOC
     I                                    P 179 1820SAFTY
     ICONTROL NS
     I                                       32  370LBDTE
     I                                       38  430LBTME
     IIDATEB  NS                                                        INVALID DATES FILE
     I                                        1   40IDPGE
     I                                        5  19 IDPRT
     I                                       20  20 IDWHS
     I                                       21  250IDPLN
     I                                       26  310IDDTE
     I                                       32  38 IDORD
     I            UDS
     I                                        1   7 SEQ
     I                                        8  10 REBLD
     I                                       11  25 PART1
     I                                       26  40 PART2
     I                                       41  41 WHSE1
     I                                       42  42 WHSE2
     I                                       43  44 CLASS1
     I                                       45  46 CLASS2
     I                                       47  47 TYPE1
     I                                       48  48 TYPE2
     I                                       49  530PLAN1
     I                                       54  580PLAN2
     I                                       61 110 SUBTTL
     C   NO3                 TIME           SYS     120
```

```
C  N03       SYS       DIV  1000000   SYSTME  60
C  N03                 MVR            SYSDTE  60
C  N03                 MOVE *BLANK    KISDB  31
C  N03                 MOVEL'C'       KISDB
C  N03       KISDB     CHAINCONTROL              90
C  N03                 EXCPTHEADER
C  N03                 SETON                     03
C   OA                 EXCPTHEADER
C   L1                 SETOF                     50
C   L1       OOH       COMP 0                      50
C   L1                 EXCPTNEW
C   L1                 Z-ADDOOH       REM     70
C   L1                 Z-ADDO         FLAG    10
C            ORTYP     COMP 'C'                      11
C            ORTYP     COMP 'M'                      12
C            ORTYP     COMP 'P'                      13
C            PROD      COMP 'P'                      14
C            PRTY      COMP *BLANK               15
C   11                 SUB  OBO       REM
C   12 14              ADD  OBO       REM
C   12N14              SUB  QBO       REM
C   13                 ADD  QBO       REM
C                      Z-ADDDUEDT     YMD     60
C                      EXSR YMDMDY
C            REM       COMP 0000000                50
C   50                 Z-ADD1         FLAG
C                      EXCPTLINE
C                      EXSR CKDATE
CL1                    EXSR TEST
CLR                    EXSR WRTEOJ
C*
C            TEST      BEGSR
C            FLAG      COMP 1                        50
C   50                 ADD  1         SHORT   50
C                      ENDSR
C*
C            WRTEOJ    BEGSR
C                      EXCPTEOJ
C            IDATES    IFGT *ZERO
C                      EXSR IOLIST
C                      END
C                      ENDSR
C*
C            YMDMDY    BEGSR
C            YMD       MULT .0001     MMMMM1  20
C            MMMMM1    MULT 10000     MMMMM2  60
C            YMD       SUB  MMMMM2    MDY     60
C                      MULT 100       MDY
C                      ADD  MMMMM1    MDY
C                      ENDSR
C*
C            CKDATE    BEGSR
C            MDY       DIV  10000     MM      20
C                      MVR            DDYY    40
C            DDYY      DIV  100       DD      20
C                      MVR            YY      20
C                      Z-ADDSYSDTE    SYSYY   20
C            SYSYY     SUB  YY        DIFFYY  20
C            DIFFYY    IFLT *ZERO
C                      Z-SUBDIFFYY    DIFFYY
C                      END
C            MM        IFGE 01
C            MM        IFLE 12
C            DD        IFGE 01
C            DD        IFLE 31
C            DIFFYY    IFLE 08
C                      GOTO RTNCK
C                      END
C                      END
C                      END
C                      END
C                      END
C                      Z-ADDPAGE      IDPGE   40
C                      EXCPTERRDTE
C                      ADD  0000001   IDATES  70
```

```
C           RTNCK      ENDSR
C*
C           IDLIST     BEGSR                              LIST INVALID DATES FOUND
C                      EXCPTIDHDR
C                      Z-ADD0000001   RRN     70
C           IDLOOP     TAG
C           RRN        CHAINIDATEB               10
C  N10 OA              EXCPTIDHDR
C  N10                 EXCPTIDREC
C  N10                 ADD  0000001   RRN
C  N10                 GOTO IDLOOP
C                      EXCPTIDBTM
C                      ENDSR
OSHRTAR  E  104            HEADER
O                          SYSDTEY    8
O                          SYSTME    18 '0  :  .  '
O                                    66 'ITEM SHORTAGE REPORT BY'
O                          SEQ       74
O                                   114 'REPTSHRT - PAGE'
O                          PAGE  3  119
O        E  2              HEADER
O                                    14 'NEW DATABASE ='
O                          REBLD     18
O                          SUBTTL    84
O                                   101 'BUILT:'
O                          LBDTE    110 '0  /  /  '
O                          LBTME    119 '0  :  :  '
O        EF 1              NEW
O                                    24 '------------'
O                          *PLACE    48
O                          *PLACE    72
O                          *PLACE    96
O                                   119 '------------'
O        E  1              NEW
O                                    17 'COMPONENT      WH'
O                                    29 'DESCRIPTION'
O                                    64 'PLANR  T CL UM'
O                                    86 'SAFETY  ALLOCATED'
O                                   108 'ORDER (M)  ORDER (P)'
O                                   119 'ON HAND'
O        E  0              NEW
O                          PART#     15
O                          PLANR     55
O        E  1              NEW
O                          PART#     15
O                          WHSE#     17
O                          DESCR     48
O                          PLANR     55
O                          TYPE      58
O                          CLASS     61
O                          UMEAS     64
O                          SAFTY J   76
O                          ALLOC J   87
O                          QOOM  J   98
O                          QOOP  J  109
O                          QOH   J  120
O              50                   128 'NEG'
O        E  1              NEW
O                                    22 'ORDER   ITEM/REF#'
O                                    54 'WH  DESCRIPTION/CUSTOMER'
O                                    86 'REQ DATE   DUE DATE'
O                                   108 'REQ QTY   RECEIPTS'
O                                   119 'REMAINING'
O        EF 1              LINE
O                          STAT       2
O              12 14 15               3 '.'
O                          PRTY       4
O                          ORDER     12
O              11          CNAME     59
O              11          MDY   Y   75
O              11          QBO   J   98
O              12 14       REF#      23
O              12 14       CJOB#     40
O              12 14       MDY   Y   86
O              12 14       QBO   J  109
```

```
O                12N14   PARNT#    28
O                12N14   WHSE#     32
O                12N14   PDESC     64
O                12N14   MDY    Y  75
O                12N14   QBO    J  98
O                13      MDY    Y  86
O                13      QBO    J 109
O                        REM    J 120
O                50               129 'SHORT'
O        E  1            ERRDTE
O                                  88 '** INVALID DATE ABOVE **'
O        EF 1            EOJ
O                                  24 '------------'
O                        *PLACE    48
O                        *PLACE    72
O                        *PLACE    96
O                                 119 '------------'
O        E               EOJ
O                        SHORT 1    6
O                                  18 'ITEMS SHORT'
O        E  306          EOJ
O                                  24 'REPORT LIMITS & OPTIONS:'
O        E 22            EOJ
O                                  24 'FROM'
O                                  42 'TO'
O        E  1            EOJ
O                                  12 'PART NUMBERS'
O                        PART1     35
O                        PART2     55
O        E  1            EOJ
O                                  10 'WAREHOUSES'
O                        WHSE1     21
O                        WHSE2     41
O        E  1            EOJ
O                                  11 'CLASS CODES'
O                        CLASS1    22
O                        CLASS2    42
O        E  1            EOJ
O                                  10 'TYPE CODES'
O                        TYPE1     21
O                        TYPE2     41
O        E  3            EOJ
O                                   8 'PLANNERS'
O                        PLAN1     25
O                        PLAN2     45
O        E  106          IDHDR                                      INVALID DATE HEADER
O                        SYSDTEY    8
O                        SYSTME    18 '0  .  :  '
O                                  66 'ITEM SHORTAGE REPORT BY'
O                        SEQ       74
O                                 114 'REPTSHRT - PAGE'
O                        PAGE  3  119
O        E  2            IDHDR
O                                  14 'NEW DATABASE ='
O                        REBLD     18
O                        SUBTTL    84
O                                 101 'BUILT:'
O                        LBDTE    110 '0  /  /  '
O                        LBTME    119 '0  :  :  '
O        E  3            IDHDR
O                                  70 '*** INVALID DATES ***'
O        E  2            IDHDR
O                                  48 'PAGE    PART NUMBER'
O                                  78 'WH#    PLANNER    ORDER #'
O                                  87 'DATE'
O        E  1            IDREC
O                        IDPGE 3   34
O                        IDPRT     52
O                        IDWHS     57
O                        IDPLN     67
O                        IDORD     78
O                        IDDTE Y   89
O        E 11            IDBTM
O                        IDATES1    9
O                                  30 'INVALID DATES FOUND.'
```

```
OIDATEA  E                 ERRDTE
O                          IDPGE      4
O                          PART#     19
O                          WHSE#     20
O                          PLANR     25
O                          MDY       31
O                          ORDER     38
```

Figure 15-13

Procedure @RPTSMPL

```
** PROC-@RPTSMPL.
**
** Copy the original source program into a DISK file, SMPL1?ws?
//   LOAD $MAINT
//     FILE NAME-SMPL1?WS?,UNIT-F1,RECORDS-256,EXTEND-128
//     RUN
//     COPY FROM-?L'7,8'?,TO-DISK,FILE-SMPL1?WS?,NAME-?L'1,6'?,LIBRARY-S,RECL-80
//     END
**
** Build a DISK file, SMPL2?ws?, containing the generated source program.
//   LOAD SMPLA@
//     FILE NAME-ORIG,LABEL-SMPL1?WS?,RETAIN-S
//     FILE NAME-SMPL,LABEL-SMPL2?WS?,RECORDS-256,EXTEND-128
//     RUN
**
** The WORKLIBR library is required so that user libraries aren't disturbed.
** If not found, build it.
//   IFF DATAF1-WORKLIBR BLDLIBR WORKLIBR,30,15
//   CONDENSE WORKLIBR
**
** Copy the SMPL2?ws? file into a source member, SMPLB@.
//   LOAD $MAINT
//     FILE NAME-SMPL2?WS?,UNIT-F1,RETAIN-S
//     RUN
//     COPY TO-WORKLIBR,FROM-DISK,FILE-SMPL2?WS?,NAME-SMPLB@,LIBRARY-S
//     END
**
** Compile the generated source program.
//   RPGC SMPLB@,WORKLIBR,,NOPRINT
**
** If compile was unsuccessful, issue a MSG and print the generated source.
** Then, remove it from WORKLIBR.
** Else, GOTO OK.
//   IF LOAD-'SMPLB@,WORKLIBR' GOTO OK
//   MSG ?WS?,REPTSMPL ABORTED.  GENERATED SOURCE HAS LOGIC ERROR.
//   MSG ?WS?,SEE PRINTOUT OF GENERATED SOURCE MEMBER.
//   LOAD $MAINT
//     RUN
//     COPY NAME-SMPLB@,LIBRARY-S,FROM-WORKLIBR,TO-PRINT
//     DELETE NAME-SMPLB@,LIBRARY-ALL,LIBRNAME-WORKLIBR
//     COMPRESS LIBRNAME-WORKLIBR
//     END
//   RETURN
// TAG OK
**
** Load and run the compiled printing program.  This prints the sample.
//   LOAD SMPLB@,WORKLIBR
//     RUN
**
** Remove both source and object (generated) members from WORKLIBR and
** condense them.  The original RPG-II source member is, of course, unaffected.
//   LOAD $MAINT
//     RUN
//     DELETE NAME-SMPLB@,LIBRARY-ALL,LIBRNAME-WORKLIBR
//     COMPRESS LIBRNAME-WORKLIBR
//     END
```

Figure 15-14

Program SMPLA@

```
*       1 .     . 2       . 3 . .     4  .     . 5       .  6  .      7 .    .  8
     H                                                                      SMPLA@
      *
      *
     FORIG    IPE F8000  80            DISK
     FSMPL    O   F8000  80            DISK
     F*INDICATOR USAGE
     F*01-07: RECORD IDENTIFYING INDICATORS FOR THE INPUT FILE.
     F*09: FIRST-RECORD PROCESSING
     F*10-11. LOKUP OP-CODE SUCCESSFUL
     F*50· FIRST C-SPEC.  USED TO BUILD E-SPECS RIGHT BEFORE IT
     F*51  FIRST O-SPEC   USED TO BUILD THE LAST C-SPECS RIGHT BEFORE IT
     F*LR· END OF PROGRAM
      *
      *
     E                  FN          8  8              PRINTER FILE NAMES
     E                  OK          6  2              VALID INDICATORS
     E                  IND         3  3              INDICATORS FOUND IN O-SPECS
     E                  W6          6  1              WORK ARRAY FOR FIELD NAMES
     E                  O15     4   4  1              VALID VALUES FOR O-SPEC COL#15
      *
      *
     IORIG    NS  01   6 CH   7NC*   1NC/                                 H-SPEC
     I                                       1   2 STARS
     I                                       6   6 SPEC
     I                                      18  18 H1818                  CURRENCY SYMBOL
     I                                      19  19 H1919                  DATE FORMAT
     I                                      20  20 H2020                  DATE EDIT
     I                                      21  21 H2121                  INVERTED PRINT
     I        NS  02   6 CF   7NC*   1NC/                                 F-SPEC
     I                                       1   2 STARS
     I                                       6   6 SPEC
     I                                       7  14 F0714                  FILE NAME
     I                                      24  27 F2427                  RECORD LENGTH
     I                                      33  34 F3334                  OVERFLOW INDICATOR
     I                                      40  46 F4046                  DEVICE
     I        NS  03   6 CE   7NC*   1NC/                                 E-SPEC
     I                                       1   2 STARS
     I                                       6   6 SPEC
     I                                      27  32 E2732                  ARRAY NAME
     I                                      40  420E4042                  LENGTH OF ELEMENT
     I                                      44  44 E4444                  DECIMAL POSITIONS
     I                                      46  51 E4651                  ARRAY NAME
     I                                      52  540E5254                  LENGTH OF ELEMENT
     I                                      56  56 E5656                  DECIMAL POSITIONS
     I        NS  04   6 CI   7NC*   1NC/                                 I-SPEC
     I                                       1   2 STARS
     I                                       6   6 SPEC
     I                                      43  43 I4343                  PACK/BINARY
     I                                      44  470I4447                  FROM POSITION
     I                                      48  510I4851                  TO POSITION
     I                                      52  52 I5252                  DECIMAL POSITIONS
     I                                      53  58 I5358                  FIELD NAME
     I        NS  05   6 CC   7NC*   1NC/                                 C-SPEC
     I                                       1   2 STARS
     I                                       6   6 SPEC
     I                                      43  48 C4348                  RESULT FIELD NAME
     I                                      49  510C4951                  FIELD LENGTH
     I                                      52  52 C5252                  DECIMAL POSITIONS
     I        NS  06   6 CO   7NC*   1NC/                                 O-SPEC
     I                                       1   2 STARS
     I                                       6   6 SPEC
     I                                       7  14 O0714                  FILE NAME
     I                                      15  15 O1515                  TYPE OF LINE (H,D,T,E)
     I                                      16  16 O1616
     I                                      17  17 O1717                  SPACE BEFORE
     I                                      18  18 O1818                  SPACE AFTER
     I                                      19  20 O1920                  SKIP BEFORE
     I                                      21  22 O2122                  SKIP AFTER
     I                                      23  31 IND
     I                                      32  37 O3237                  FIELD/EXCPT NAME
```

```
I                                         38  38 03838               EDIT CODE
I                                         40  43 04043               FIELD NAME
I                                         45  70 04570               CONSTANT/EDIT WORD
I        NS  07                                                      ANYTHING ELSE
I                                          1   2 STARS
I                                          6   6 SPEC
I          UDS
I                                          1   6 PRGM                ORIGINAL PROGRAM
I                                          7  14 LIBR                ORIGINAL LIBRARY
I                                         15  26 OK                  INDICATORS TO SETON
 *
 *
C*BUILD A "// COPY NAME-SMPLB@,LIBRARY-S" AS THE FIRST RECORD OF
C*OUR OUTPUT FILE.  THIS IS REQUIRED BY $MAINT IN ORDER TO COPY
C*THE DISK FILE INTO A LIBRARY MEMBER.
C  NO9                 EXCPTFIRST
C  NO9                 SETON                     09
C*IF "**" FOUND IN COLUMNS 1-2 ANYTIME, COMPILE-TIME DATA FOLLOWS
C*IN THE ORIGINAL PROGRAM.  STOP PROCESSING.
C           STARS      COMP '**'                   LR
C   LR                 GOTO END
C*BRANCH TO AN APPROPRIATE SUBROUTINE ACCORDING TO THE SPECIFICATION
C*TYPE, SINCE EACH SPECIFICATION REQUIRES DIFFERENT PROCESSING.
C           SPEC       CASEQ'H'       HSPEC
C           SPEC       CASEQ'F'       FSPEC
C           SPEC       CASEQ'E'       ESPEC
C           SPEC       CASEQ'I'       ISPEC
C           SPEC       CASEQ'C'       CSPEC
C           SPEC       CASEQ'O'       OSPEC
C                      END
C           END        TAG
C*BUILD A "// CEND" RECORD AS THE LAST RECORD OF THE OUTPUT FILE.
C*THIS IS REQUIRED BY $MAINT.
CLR                    EXCPTLAST
C*
C           HSPEC      BEGSR                             COPY THE ORIGINAL H-SPEC
C                      EXCPTWRTH
C                      Z-ADD01        I        20        PRINTER FILE COUNTER INITIALIZED
C                      ENDSR
C*
C           FSPEC      BEGSR
C           F4046      IFEQ 'PRINTER'                    PROCESS ONLY IF PRINTER FILE USED
C           I          IFLE 08                           ...AND WHILE I<=8
C                      MOVE F0714     FN,I               PUT FILENAME IN ARRAY ELEMENT
C                      EXCPTWRTF                         BUILD THE F-SPEC
C                      ADD  01        I
C                      END
C                      END
C                      ENDSR
C*
C           ESPEC      BEGSR
C                      MOVE E2732     FLDNM  6           ASSUME FIELDNAME = ARRAYNAME
C                      Z-ADDE4042     FLEN   30          FIELDLENGTH = ELEMENTLENGTH
C                      MOVE E4444     DECIM  1
C                      MOVE 'N'       TYPE   1           NOT PACKED, NOT BINARY
C                      EXSR BLDC                         BUILD A C-SPEC
C                      MOVE E4651     FLDNM              SECONDARY ARRAY (SAME PROCESSING)
C                      Z-ADDE5254     FLEN
C                      MOVE E5656     DECIM
C                      MOVE 'N'       TYPE
C                      EXSR BLDC
C                      ENDSR
C*
C           ISPEC      BEGSR
C                      MOVE I4343     TYPE               UNPACKED/PACKED/BINARY
C           I4851      SUB  I4447     FLEN               CALCULATE FIELD LENGTH
C                      ADD  001       FLEN
C                      MOVE I5252     DECIM
C                      MOVE I5358     FLDNM
C                      EXSR BLDC                         BUILD A C-SPEC
C                      ENDSR
C*
C           CSPEC      BEGSR
C                      MOVE 'N'       TYPE               FOR EACH RESULTING FIELD...
C                      Z-ADDC4951     FLEN
C                      MOVE C5252     DECIM
```

```
C                 MOVE C4348     FLDNM              . .BUILD A C-SPEC
C                 EXSR BLDC
C                 ENDSR
C*
C         OSPEC   BEGSR
C  N51            EXSR LASTC                        BUILD THE LAST C-SPECS
C  N51            SETON                       51    IF FIRST O-SPEC FOUND.
C         00714   IFNE *BLANK                       IF FILENAME NOT BLANK...
C         00714   IFNE '       A'                   ...AND NOT AN "AND" LINE...
C         00714   IFNE '       O'                   ...AND NOT AN "OR" LINE...
C                 MOVE 00714     CURRFN  8          ...MOVE FILENAME TO "CURRENT FILE"
C                 END
C                 END
C                 END
C         01515   LOKUPO15                      11
C   11            MOVE 'E'       01515              MAKE IT AN EXCEPTION OUTPUT LINE
C         01515   IFNE *BLANK
C                 MOVE *BLANK    03237              ERASE EXCPT NAME
C                 END
C                 Z-ADDO         ERROR  10          SEE IF INDICATORS MATCH THOSE
C         01      DO   03        I                  SELECTED BY THE USER.
C                 MOVE IND,I     INDIC   2
C         INDIC   IFNE *BLANK
C         INDIC   LOKUPOK                       12
C  N12            ADD  1         ERROR
C                 END
C                 END
C         ERROR   IFEQ 0                            IF INDICATORS MATCH...
C         CURRFN  LOKUPFN                       10...AND FILENAME IS PRINTER FILE...
C   10            EXCPTWRTO                         ...THEN BUILD THE O-SPEC
C                 END
C                 ENDSR
C*
C         BLDC    BEGSR
C  N50            EXCPTC1                           BUILD E-SPECS AND DO-LOOPS IF NO
C  N50            SETON                       50    C-SPECS WERE BUILT YET.
C         FLEN    IFGT 000                          PROCESS ONLY IF FIELD LENGTH > 0
C         FLDNM   IFNE *BLANK                       ...AND IF FIELD NAME ISN'T BLANK
C         TYPE    IFEQ 'P'                          IF PACKED, RECALCULATE FIELD LENGTH
C                 MULT 002       FLEN               AS DOUBLE THE ORIGINAL...
C                 SUB  001       FLEN               ...MINUS 1
C                 ELSE
C         TYPE    IFEQ 'B'                          IF BINARY, RECALCULATE FIELD LENGTH
C         FLEN    IFEQ 002                          IF 2, MAKE IT 4
C                 Z-ADD004       FLEN
C                 ELSE
C         FLEN    IFEQ 004                          IF 4, MAKE IT 9
C                 Z-ADD009       FLEN
C                 END
C                 END
C                 END
C                 END
C                 MOVEAFLDNM     W6                 REMOVE ARRAY COMMA & INDEX
C         1       DO   6         J      10          IF FOUND ANYWHERE WITHIN THE
C         W6,J    IFEQ ','                          FIELD NAME.
C         J       DO   6         JJ     10
C                 MOVE ' '       W6,JJ
C                 END
C                 END
C                 END
C                 MOVEAW6        FLDNM
C         DECIM   IFEQ *BLANK                       IF ALPHAMERIC FIELD
C                 EXCPTWRTCX                        ...USE THE ALLX ARRAY
C                 ELSE
C                 EXCPTWRTC9                        IF NUMERIC, USE THE ALL9 ARRAY.
C                 END
C                 END
C                 END
C                 ENDSR
C*
C         LASTC   BEGSR
C         01      DO   06        I                  GENERATE A SETON C-SPEC FOR EACH
C         OK,I    IFNE *BLANK                       INDICATOR SELECTED BY USER
C                 EXCPTSETON
C                 END
C                 END
```

```
C                     EXCPTCLR                          GENERATE EXCPT & SETON-LR LINES
C                     ENDSR
 *
 *
OSMPL    E                 FIRST
O                                   24 '// COPY NAME-SMPLB@.LIBR'
O                                   29 'ARY-S'
O        E                 WRTH
O                                    6 'H'
O                          H1818    18
O                          H1919    19
O                          H2020    20
O                          H2121    21
O                                   80 'SMPLB@'
O        E                 WRTF
O                                    6 'F'
O                          F0714    14
O                                   19 'O    F'
O                          F2427    23
O                          F2427    27
O                          F3334    34
O                                   46 'PRINTER'
O        E                 C1
O                                    6 'E'
O                                   42 'ALLX        256   1'
O        E                 C1
O                                    6 'E'
O                                   42 'ALL9         15   1'
O        E                 C1
O                                    6 'C'
O                                   20 '001'
O                                   35 'DO    256'
O                                   52 'I          30'
O        E                 C1
O                                    6 'C'
O                                   31 'MOVE'
O                                   35 '''X'''
O                                   48 'ALLX,I'
O        E                 C1
O                                    6 'C'
O                                   30 'END'
O        E                 C1
O                                    6 'C'
O                                   20 '001'
O                                   35 'DO    015'
O                                   43 'I'
O        E                 C1
O                                    6 'C'
O                                   31 'MOVE'
O                                   35 '''9'''
O                                   48 'ALL9,I'
O        E                 C1
O                                    6 'C'
O                                   30 'END'
O        E                 WRTCX
O                                    6 'C'
O                                   36 'MOVEAALLX'
O                          FLDNM    48
O                          FLEN     51
O        E                 WRTC9
O                                    6 'C'
O                                   36 'MOVEAALL9'
O                          FLDNM    48
O                          FLEN     51
O                          DECIM    52
O        E                 WRTC9
O                                    6 'C'
O                                   32 'Z-SUB'                         FORCE NEGATIVE SIGN
O                          FLDNM    38
O                          FLDNM    48
O        E                 SETON
O                                    6 'C'
O                                   32 'SETON'
O                          OK,I     55
O        E                 CLR
O                                    6 'C'
```

```
     O                                  32 'EXCPT'
     O        E                CLR
     O                                   6 'C'
     O                                  32 'SETON'
     O                                  55 'LR'
     O        E                WRTO
     O                                   6 '0'
     O                         00714    14
     O                         01515    15
     O                         01616    16
     O                         01717    17
     O                         01818    18
     O                         01920    20
     O                         02122    22
     O                         IND      31
     O                         03237    37
     O                         03838    38
     O                         04043    43
     O                         04570    70
     O        E                LAST
     O                                   6 '0'
     O                                  20 'E  150'
     O        E                LAST
     O                                   6 '0'
     O                                  43 '21'
     O                                  45 ''''
     O                                  67 '*** END OF SAMPLE ***'''
     O        E                LAST
     O                                   6 '0'
     O                                  18 'E  1'
     O        E                LAST
     O                                   6 '0'
     O                                  43 '17'
     O                                  45 ''''
     O                         LIBR     53
     O                         PRGM     60
     O                                  61 ''''
     O        E                LAST
     O                                   7 '// CEND'
**       015
DEHT
```

Figure 15-15

Program SMPLB@

```
*  .  . 1        . 2         . 3 . .  . 4     . . . 5 . . . .  6 . . . . . . 7  .  . . . 8
H                                                                    SMPLB@
FSHRTAR  O   F 132 132       OA      PRINTER
E                    ALLX      256  1
E                    ALL9       15  1
C           001       DO   256       I       30
C                     MOVE 'X'       ALLX,I
C                     END
C           001       DO   015       I
C                     MOVE '9'       ALL9,I
C                     END
C                     MOVEAALLX      PART# 015
C                     MOVEAALLX      WHSE# 001
C                     MOVEAALLX      PLANN 003
C                     MOVEAALL9      DUEDT 0060
C                     Z-SUBDUEDT     DUEDT
C                     MOVEAALLX      PROD  001
C                     MOVEAALLX      ORTYP 001
C                     MOVEAALLX      ORDER 007
C                     MOVEAALLX      DESCR 030
C                     MOVEAALLX      PARNT#015
C                     MOVEAALLX      PDESC 030
C                     MOVEAALLX      TYPE  001
C                     MOVEAALLX      CLASS 002
C                     MOVEAALL9      PLANR 0050
C                     Z-SUBPLANR     PLANR
C                     MOVEAALLX      CNAME 025
C                     MOVEAALLX      PRTY  001
C                     MOVEAALLX      UMEAS 002
C                     MOVEAALLX      STAT  002
C                     MOVEAALLX      REF#  010
```

```
C                     MOVEAALLX     CJOB# 006
C                     MOVEAALL9     QBO   0070
C                     Z-SUBQBO      QBO
C                     MOVEAALL9     QOH   0070
C                     Z-SUBQOH      QOH
C                     MOVEAALL9     QOOM  0070
C                     Z-SUBQOOM     QOOM
C                     MOVEAALL9     QOOP  0070
C                     Z-SUBQOOP     QOOP
C                     MOVEAALL9     ALLOC 0070
C                     Z-SUBALLOC    ALLOC
C                     MOVEAALL9     SAFTY 0070
C                     Z-SUBSAFTY    SAFTY
C                     MOVEAALL9     LBDTE 0060
C                     Z-SUBLBDTE    LBDTE
C                     MOVEAALL9     LBTME 0060
C                     Z-SUBLBTME    LBTME
C                     MOVEAALL9     IDPGE 0040
C                     Z-SUBIDPGE    IDPGE
C                     MOVEAALLX     IDPRT 015
C                     MOVEAALLX     IDWHS 001
C                     MOVEAALL9     IDPLN 0050
C                     Z-SUBIDPLN    IDPLN
C                     MOVEAALL9     IDDTE 0060
C                     Z-SUBIDDTE    IDDTE
C                     MOVEAALLX     IDORD 007
C                     MOVEAALLX     SEQ   007
C                     MOVEAALLX     REBLD 003
C                     MOVEAALLX     PART1 015
C                     MOVEAALLX     PART2 015
C                     MOVEAALLX     WHSE1 001
C                     MOVEAALLX     WHSE2 001
C                     MOVEAALLX     CLASS1002
C                     MOVEAALLX     CLASS2002
C                     MOVEAALLX     TYPE1 001
C                     MOVEAALLX     TYPE2 001
C                     MOVEAALL9     PLAN1 0050
C                     Z-SUBPLAN1    PLAN1
C                     MOVEAALL9     PLAN2 0050
C                     Z-SUBPLAN2    PLAN2
C                     MOVEAALLX     SUBTTL050
C                     MOVEAALL9     SYS   0120
C                     Z-SUBSYS      SYS
C                     MOVEAALL9     SYSTME0060
C                     Z-SUBSYSTME   SYSTME
C                     MOVEAALL9     SYSDTE0060
C                     Z-SUBSYSDTE   SYSDTE
C                     MOVEAALLX     KISDB 031
C                     MOVEAALL9     REM   0070
C                     Z-SUBREM      REM
C                     MOVEAALL9     FLAG  0010
C                     Z-SUBFLAG     FLAG
C                     MOVEAALL9     YMD   0060
C                     Z-SUBYMD      YMD
C                     MOVEAALL9     SHORT 0050
C                     Z-SUBSHORT    SHORT
C                     MOVEAALL9     MMMMM10020
C                     Z-SUBMMMMM1   MMMMM1
C                     MOVEAALL9     MMMMM20060
C                     Z-SUBMMMMM2   MMMMM2
C                     MOVEAALL9     MDY   0060
C                     Z-SUBMDY      MDY
C                     MOVEAALL9     MM    0020
C                     Z-SUBMM       MM
C                     MOVEAALL9     DDYY  0040
C                     Z-SUBDDYY     DDYY
C                     MOVEAALL9     DD    0020
C                     Z-SUBDD       DD
C                     MOVEAALL9     YY    0020
C                     Z-SUBYY       YY
C                     MOVEAALL9     SYSYY 0020
C                     Z-SUBSYSYY    SYSYY
C                     MOVEAALL9     DIFFYY0020
C                     Z-SUBDIFFYY   DIFFYY
C                     MOVEAALL9     IDPGE 0040
```

```
C                    Z-SUBIDPGE     IDPGE
C                    MOVEAALL9      IDATES0070
C                    Z-SUBIDATES    IDATES
C                    MOVEAALL9      RRN    0070
C                    Z-SUBRRN       RRN
C                    SETON                     12
C                    SETON                     14
C                    SETON                     50
C                    EXCPT
C                    SETON                     LR
OSHRTAR  E  104
O                        SYSDTEY   8
O                        SYSTME   18 '0  :  :  '
O                                 66 'ITEM SHORTAGE REPORT BY'
O                        SEQ      74
O                                114 'REPTSHRT - PAGE'
O                        PAGE  3 119
O        E  2
O                                 14 'NEW DATABASE -'
O                        REBLD    18
O                        SUBTTL   84
O                                101 'BUILT:'
O                        LBDTE   110 '0  /  /  '
O                        LBTME   119 '0  :     '
O        EF 1
O                                 24 '------------'
O                        *PLACE   48
O                        *PLACE   72
O                        *PLACE   96
O                                119 '------------'
O        E  1
O                                 17 'COMPONENT      WH'
O                                 29 'DESCRIPTION'
O                                 64 'PLANR  T CL UM'
O                                 86 'SAFETY  ALLOCATED'
O                                108 'ORDER (M)  ORDER (P)'
O                                119 'ON HAND'
O        E  0
O                        PART#    15
O                        PLANR    55
O        E  1
O                        PART#    15
O                        WHSE#    17
O                        DESCR    48
O                        PLANR    55
O                        TYPE     58
O                        CLASS    61
O                        UMEAS    64
O                        SAFTY J  76
O                        ALLOC J  87
O                        QOOM  J  98
O                        QOOP  J 109
O                        QOH   J 120
O                50              128 'NEG'
O        E  1
O                                 22 'ORDER   ITEM/REF#'
O                                 54 'WH  DESCRIPTION/CUSTOMER'
O                                 86 'REQ DATE   DUE DATE'
O                                108 'REQ QTY   RECEIPTS'
O                                119 'REMAINING'
O        EF 1
O                        STAT      2
O                        PRTY      4
O                        ORDER    12
O                12 14   REF#     23
O                12 14   CJOB#    40
O                12 14   MDY   Y  86
O                12 14   QBO   J 109
O                12N14   PARNT#   28
O                12N14   WHSE#    32
O                12N14   PDESC    64
O                12N14   MDY   Y  75
O                12N14   QBO   J  98
O                        REM   J 120
O                50              129 'SHORT'
```

```
     O        E  1
     O                                        88 '** INVALID DATE ABOVE **'
     O        EF 1
     O                                        24 '------------'
     O                              *PLACE    48
     O                              *PLACE    72
     O                              *PLACE    96
     O                                       119 '------------'
     O        E
     O                              SHORT 1    6
     O                                        18 'ITEMS SHORT'
     O        E  306
     O                                        24 'REPORT LIMITS & OPTIONS:'
     O        E 22
     O                                        24 'FROM'
     O                                        42 'TO'
     O        E  1
     O                                        12 'PART NUMBERS'
     O                              PART1     35
     O                              PART2     55
     O        E  1
     O                                        10 'WAREHOUSES'
     O                              WHSE1     21
     O                              WHSE2     41
     O        E  1
     O                                        11 'CLASS CODES'
     O                              CLASS1    22
     O                              CLASS2    42
     O        E  1
     O                                        10 'TYPE CODES'
     O                              TYPE1     21
     O                              TYPE2     41
     O        E  3
     O                                         8 'PLANNERS'
     O                              PLAN1     25
     O                              PLAN2     45
     O        E  106
     O                              SYSDTEY    8
     O                              SYSTME    18 '0  .  .  '
     O                                        66 'ITEM SHORTAGE REPORT BY'
     O                              SEQ       74
     O                                       114 'REPTSHRT - PAGE'
     O                              PAGE  3  119
     O        E  2
     O                                        14 'NEW DATABASE -'
     O                              REBLD     18
     O                              SUBTTL    84
     O                                       101 'BUILT:'
     O                              LBDTE    110 '0  /  /  '
     O                              LBTME    119 '0  :  :  '
     O        E  3
     O                                        70 '*** INVALID DATES ***'
     O        E  2
     O                                        48 'PAGE    PART NUMBER'
     O                                        78 'WH#    PLANNER    ORDER #'
     O                                        87 'DATE'
     O        E  1
     O                              IDPGE 3   34
     O                              IDPRT     52
     O                              IDWHS     57
     O                              IDPLN     67
     O                              IDORD     78
     O                              IDDTE Y   89
     O        E 11
     O                              IDATES1    9
     O                                        30 'INVALID DATES FOUND.'
     O        E  150
     O                                        21 '*** END OF SAMPLE ***'
     O        E  1
     O                                        17 'CODELIBR SHRTAR'
```

Figure 15-16

Sample of an item shortage report

```
99/99/99  99:99:99                  ITEM SHORTAGE REPORT BY XXXXXXX                 REPTSHRT - PAGE    1
NEW DATABASE - XXX             XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX        BUILT: 99/99/99 99:99:99
-----------------------------------------------------------------------------------------------------------
COMPONENT      WH DESCRIPTION                     PLANR  T CL UM     SAFETY  ALLOCATED  ORDER (M)  ORDER (P)    ON HAND
XXXXXXXXXXXXXXX                                   9999R
XXXXXXXXXXXXXXX X XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  9999R  X XX XX  9,999,999- 9,999,999- 9,999,999- 9,999,999- 9,999,999-     NEG
     ORDER   ITEM/REF#        WH  DESCRIPTION/CUSTOMER              REQ DATE   DUE DATE    REQ QTY   RECEIPTS  REMAINING
XX X XXXXXXX XXXXXXXXXX           XXXXXX                                       99/99/99              9,999,999- 9,999,999-    SHORT
                                                                 ** INVALID DATE ABOVE **
-----------------------------------------------------------------------------------------------------------
99,999 ITEMS SHORT


REPORT LIMITS & OPTIONS:
                    FROM                TO
PART NUMBERS        XXXXXXXXXXXXXXX     XXXXXXXXXXXXXXX
WAREHOUSES          X                   X
CLASS CODES         XX                  XX
TYPE CODES          X                   X
PLANNERS            9999R               9999R


                                                       *** INVALID DATES ***
                              PAGE   PART NUMBER      WH#   PLANNER   ORDER #     DATE
                              9999   XXXXXXXXXXXXXXX   X     9999R    XXXXXXX   99/99/99
9,999,999 INVALID DATES FOUND.
*** END OF SAMPLE ***
  CODELIBR SHRTAR
```

Printing Tips for Hold, Halt, Align

by Jerry Inhoff

I have found that many users and programmers forget some of the simple points they learned early in their computing experience. With that in mind, I offer three printing tips.

If you have any print jobs that you would like to hold and print later, use the PRINTER OCL statement. If you specify PRIORITY-0, your entry will be placed on the spool file with a priority of 1, but it will be held. Such an entry is printed when a RELEASE control command specifically indicates that it is to be printed.

If any of your programs halt after printing a line that contains an unprintable character, place a 1 in position 45 of the H-spec of the program that produces the report. (Don't forget to recompile the program after the change is made.) Then, when an unprintable character is encountered, it is replaced with a blank, but no program halt occurs. Please note that, because the unprintable character is not printed, your output will be incorrect. Make certain your operators know to find the blank and fill it in manually with the appropriate character.

To make forms alignment easier, place a 1 in position 41 of the H-spec of the program that produces the report. If a 1 is specified, the system prints the first line of output and issues a message. You then can either realign the forms and select the option to try printing the line again, or you can select the option to continue printing if the forms are aligned. This forms specification is valid for spooled or unspooled output, but it works only if the output is conditioned by 1P (first page indicator).

Controlling the Spool File with OCL

answered by John Fruetel

Q I need an OCL procedure to run system console commands such as HOLD and RELEASE. The OCL procedure should also allow us to start a spool writer. Can it be done?

A Yes and no. With the advent of SSP Release 5.1, four commands that previously could be run only from the system console were added to OCL:

```
// CANCEL
// CHANGE
// START
// STOP
```

These new OCL statements apply only to the spool writer. They do not allow for the manipulation of the job queue or currently running jobs. The statements also have no provision for holding and releasing spooled entries. Still, the new commands are a welcome addition to OCL.

Prompting for Report Parameters

by Joe Medeiros

Code on diskette:
Procedure REPORT
Screen format member REPORTFM

Once in a while, S/36 users probably ask you if it would be possible (and they know it is) to get an extra copy of a certain report. Obligingly, you change the number of copies specified in the reporting procedure's PRINTER statement, but before long, those same users no longer want extra copies.

Because of such users, I wrote procedure REPORT (Figure 15-17) and designed its accompanying prompt screen (see Figure 15-18 for screen format member REPORTFM). I call procedure REPORT from another procedure (see Figure 19 for a sample calling procedure). Procedure REPORT formats the system time and date for the display and then prompts the user for the number of copies and the printer to be used, and asks whether the report should be held on the spool file (i.e., PRIORITY = 0) and whether the job should be evoked. After the user answers the prompts, the procedure edits the input for errors. If no errors are found, all the parameters are passed to the calling procedure, and the parameters for the number of copies, printer ID, and the spooled output's priority are used in the // PRINTER statement.

Note that I use the LIBRARY parameter when I call procedure REPORT and in the // PROMPT statement so that I can store this utility in only one library — TOOLBOX.

Figure 15-17

Procedure REPORT

```
// LOCAL OFFSET-400,DATA-'?DATE??TIME?'
// EVALUATE P1-'?L'406,2'? ?L'408,2'? ?L'410,2'?'
// EVALUATE P2-'?L'400,2'?/?L'402,2'?/?L'404,2'?'
// TAG PROMPT
// PROMPT MEMBER-REPORTFM,FORMAT-REPORT01,LIBRARY-TOOLBOX,LENGTH-'8,8,2,2,1,1,40'
// IF ?CD?-2007 CANCEL
// IF  ?03?-00            GOTO PROMPT ?07F' Invalid number of copies '?
// IFF ?04?-P1 IFF ?04?-P2 GOTO PROMPT ?07F' Invalid printer selected '?
// IFF ?05?-Y  IFF ?05?-N  GOTO PROMPT ?07F' Hold report.     Y or N ?'?
// IFF ?06?-Y  IFF ?06?-N  GOTO PROMPT ?07F' Evoke this job.  Y or N ?'?
// IF  ?05?-Y  EVALUATE P05-'0'
// ELSE        EVALUATE P05-'1'
// RETURN *ALL
**___________________________________________________________________**
*  P01 - Time                           This procedure can be used     *
*  P02 - Date                           to set up any report, look     *
*  P03 - Number of copies               out for the parameters used    *
*  P04 - Printer to be used             and the LDA positions used     *
*  P05 - Place report on hold (Y-N)                                    *
*  P06 - Evoke this job (Y-N)                                          *
*  P07 - Error message                                                 *
**___________________________________________________________________**
```

Figure 15-18

Screen format member REPORTFM

```
*        1          2          3          4          5          6          7          8
0001 SREPORT01  0124          YY                                            G
0002 OTIME       8010201 Y           Y
0003 O          240129Y               Y                    CREPORT OPTIONS SELECTIOX
0004 DN
0005 DDATE       8017202 Y           Y
0006 O          270428Y                                    CNumber of copies       X
0007 D                                                               .
0008 DCOPIES     2045403 YO    ZY     Y        Y            01
0009 D          270826Y                                    CPrinter to be used     X
0010 D
0011 OPRINTER    2065404 Y            Y        Y            P2
0012 O          270828Y                                    CPlace report on hold  .X
0013 O
0014 DHOLD       1085505 YA           Y        Y            N
0015 D          271028Y                                    CEvoke this job         X
0016 O
0017 DEVOKE      1105506 YA           Y        Y            N
0018 OMSG       40192107                     07
0019 D          202302Y               Y                    C* Cmd-7 Cancel Job *
```

Figure 15-19

Sample calling procedure

```
** ORD001 - List customer master file
______________________________________________________________
// IF EVOKED-NO IF JOB0-NO REPORT,TOOLBOX
// IF EVOKED-NO IF JOB0-NO IF ?6?-Y EVOKE OR0001 *ALL
// IF EVOKED-NO IF JOB0-NO IF ?6?-Y RETURN
*_____________________________________________________________
// LOAD ORD001
// FILE NAME-CUSTMSTR,LABEL-OR,CUSTM,DISP-SHR
// PRINTER NAME-OR0001,COPIES-?3'1'?,DEVICE-?4'P1'?,PRIORITY-?5'1'?
// RUN
***********************************
```

Changing LPI, CPI, and LPP After Reports Are Created

by Joe Madeiros

Does the following scenario sound familiar? Your job has finally run successfully. The report has been placed on the print spool file. Just as you begin to relax a bit, you realize that the lines-per-form, lines-per-inch, or characters-per-inch printer control data are incorrect. Don't panic. You don't have to rerun the job; all you have to do is modify the appropriate fields in the data file header record. Simply use COPYPRT to copy the spool file to a data file, change the data file via your favorite file editor or with a file update program, use COPYPRT to copy the updated file back to the print spool file, and delete the old entry.

For example, I recently needed to change the lines per form from 20 to 30. On the spool file header record, lines per form appears as a binary value in record positions 74 and 75, so I updated this field to binary 0001 1110 (the binary equivalent of the decimal number 30). The same idea applies to the lines-per-inch and characters-per-inch values, which appear in positions 84 and 85, respectively.

Another printer control "trick" is to create one spool file entry for the reports produced by individual procedures within a job. That way you can keep track of a given job's output more easily, and if multiple copies are being printed, the multiple copies will be collated. Generally, each individual procedure produces its own spool file entry, but you can "run them together" by specifying

```
// PRINTER CONTINUE-YES
```

in the first calling procedure. To avoid possible conflicts, you may not want to use this CONTINUE statement if the individual procedures have their own PRINTER statements.

Changing CPI After a Report Is Created

by Roger Washburn

Code on diskette:
Procedure PRINT198

Many S/36 users still like to print an occasional DFU listing wider than 132 characters. And some programmers like to code RPG II printouts wider than 132 but forget to change the OCL printer statement to CPI-15. With Release 5.0, users could respond to the SYS-6151 message and temporarily change the session CPI to 15 to print the listing. With Release 5.1, however, no message is generated to the user. The job terminates normally without any error messages except for the SYS-6303 message (i.e., system error occurred while using printer *xx*) the system console receives; unfortunately, the available responses to the message do not include a print option.

After some thought and discussion, I came up with procedure PRINT198 to print the "locked up" listing and to clear it from the print spool (see Figure 15-20). Whenever the SYS-6303 message hits the system console (or sub-console), take an option 0 to put the print job on hold. Then key in

```
PRINT198 SPxxxx
```

and the printout will be generated through COPYPRT.

Figure 15-20

Procedure PRINT198

```
// IF ?1?/                 * 'Enter Spool-ID to be printed'
// IF DATAF1-?1??WS?       DELETE ?1??WS?,F1
COPYPRT ?1R?,?1??WS?,CANCEL
// PRINTER CONTINUE-YES,CPI-15
COPYPRT NOCOPY,?1??WS?,,PRINT
// PRINTER CONTINUE-YES,CPI-10
DELETE ?1??WS?,F1
```

Setting CPI and FONT for a Printer File

by George Applegate

We have a 5225 printer and a 4224 printer attached to our S/36. The 5225 requires a CPI parameter on the PRINTER statement, while the 4224 requires a FONT parameter. Unfortunately, I have some reports that I sometimes want to print on both printers, and SSP will not allow both the FONT and CPI parameters on the same PRINTER statement.

I've solved this problem by using the FORMS and PRINTER statements together to allow the report to be printed on either device:

```
// FORMS CPI-15
// LOAD PROG
// PRINTER NAME-PRINT,FONT-DF
// RUN
// FORMS CPI-10
```

Processing COPYPRT Files from a Program

answered by Ron Elliot and Mike Patton

Q Is there any way that a S/36 program can access the spool ID for a printer file (or files) that it creates? Also, is it possible to access the number of pages in a printer file? I would like to code procedures that COPYPRT automatically certain printer files and then print only the first or last page of the report.

A A program cannot access the spool files that it creates (until someone creates a patch to do so), but the next step in the procedure can. To avoid having to specify the spool ID (which would be unknown), place a // PRINTER statement in the procedure, specifying some unique FORMSNO. The value so specified can then become the first parameter of a subsequent COPYPRT statement. So, your procedure would look like this:

```
// LOAD -
.
.
.
// PRINTER NAME-PRINT,FORMSNO-WXYZ,PRIORITY-0
// RUN
COPYPRT FWXYZ,XXXX,CANCEL
```

Then disk file *XXXX* will contain the spooled print data. File *XXXX* has an informational record as the first record of the file, coded with an H (for header) in column 1. This H record contains the number of pages for the print in positions 65 and 66. (This value is represented as a binary number, and the program that processes it will have to be coded accordingly.) Hence, after the COPYPRT, you can run a program to process the file in any desired fashion.

There will be one H record for each spool entry that is COPYPRTed. The format of pertinent data in this record is as follows:

Position	Contents
4-9	Spool ID
12-19	First level procedure name
22-29	Jobname (WSHHMMSS)
42-49	Printer file name
52-53	Printer ID
56-59	Form number
65-66	Number of pages in this print file (BINARY)
69-72	Total number of records in this entry (BINARY)
74-75	Lines per page value (BINARY)

For all other records in the file, the data is as follows:

1-2	Page number within this print file (BINARY)
3-4	Line number on which to print (within this page) (BINARY)
5-8	Record number within this print file (BINARY)
11 on	Print data

Suppressing PRINT Key Output

answered by Mike Patton

Q Is there a way to cancel the PRINT key function on the S/36 so that, when users hit the darn thing by accident (as they so often do), no action takes place? IBM suggests we turn the printer off, but that solution prevents the printing of desired reports.

A On the S/36, you can disable the PRINT key during the execution of a program by adding the following OCL statement to the procedure that calls that program:

```
// WORKSTN UNIT-?WS?,PRINT-NO
```

For those cases in which no program is running, disabling the print key is a bit more problematic. I would not suggest that you turn your printer off. Instead, you can configure a "ghost" printer on your system (i.e., one that does not physically exist) and then use the PRINTKEY procedure to direct all PRINT key output to this nonexistent printer. For example:

```
PRINTKEY P0
```

would direct PRINT key output to a previously configured but nonexistent printer P0. This step eliminates unwanted paper, although PRINT key output continues to occupy spool file space on disk. Remember to delete all spool file entries for printer P0 periodically.

Resetting Forms Types for Printing After IPL

by Mel Beckman

program by Jorge Rodriquez

Code on diskette:
Procedure PRINTS
RPG program PRINTS
Message members MSG1404, MSG1404N

To give S/36 users increased access to their computer systems, DP managers typically schedule daily dedicated operations during nonprime-time hours. A common practice involves backing up files at the end of the work day and then performing an IPL that runs time-consuming keysorts. Once the IPL has begun, the system operator can go home for the evening. After the IPL is completed, the system is ready for unattended use by the late evening crowd or early morning "power users."

Yet this practice can lead to problems in shops that permanently assign different form types to individual printers. During IPL, the system "forgets" which forms are installed on each printer, and a forms-change message appears at the system console the first time a user tries to print something. If the system console is signed off because the system operator has gone home, the unhappy user is unable to sign on to the console and answer the message. The user consequently cannot get the needed printout until the start of the next workday. Fortunately, a combination of common programming techniques can automatically re-initialize your printers after every IPL.

The RPG program PRINTS (Figure 15-21) contains multiple print files that combine with procedure PRINTS (Figure 15-22) to access SSP's autoresponse capability. Just call procedure PRINTS from your S/36 #STRTUP2 procedure. Procedure PRINTS first runs the IBM-supplied RESPONSE procedure to install an autoresponse value of 1 for the forms-change message, SYS-1404 (Figure 15-23). A // NOHALT statement must follow the RESPONSE procedure immediately to enable autoresponse of system console messages.

Next, procedure PRINTS loads and runs program PRINTS. In the procedure, code a // PRINTER statement for each printer you want to set up, with the NAME parameter set to PRINT1, PRINT2, and so forth. In addition, set the FORMSNO parameter to the desired forms name. Finally, code a // SWITCH statement to turn on the UPSI switch that corresponds to each printer file (e.g., switch 1 for file PRINT1).

The program PRINTS is set up to contain eight print files, conditioned on UPSI switches U1 through U8. If you have more than eight printers, you simply execute the PRINTS program again. When program PRINTS is run, it outputs one blank line to each print file for which the associated UPSI switch is set. The spool file thus contains a blank one-page "report" for each printer.

As the SSP initiates spool writers to print these reports, it sends forms-change messages (SYS-1404) to the system console. These messages are answered automatically using the autoresponse value of 1, designated earlier by the RESPONSE procedure. After each printer prints a blank page, the printer is ready for use without operator intervention.

After program PRINTS has done its job, procedure PRINTS is called again to execute a // NOHALT statement. At this time, it also executes the RESPONSE procedure to return message handling for SYS-1404 to the operator (Figure 15-24). The automatic response is thereby inactive at times when manual response might be desirable.

With the PRINTS utility, you can be sure that forms-change messages won't "gum up the works" when you're running your S/36 in unattended mode. In addition, PRINTS can be used to set the FONT number for 5219 printers. You also may find the technique of combining an RPG program with SSP's autoresponse capability useful for controlling other unattended operations in your never-ending quest for efficient system use.

Figure 15-21

Program PRINTS

```
*        1          2          3          4          5          6          7 .        8
H
FPRINT1  O     132 132            PRINTER                               U1
FPRINT2  O     132 132            PRINTER                               U2
FPRINT3  O     132 132            PRINTER                               U3
FPRINT4  O     132 132            PRINTER                               U4
FPRINT5  O     132 132            PRINTER                               U5
FPRINT6  O     132 132            PRINTER                               U6
FPRINT7  O     132 132            PRINTER                               U7
FPRINT8  O     132 132            PRINTER                               U8
C                    SETON                     LR
OPRINT1  D        U1
```

```
0012 OPRINT2  D        U2
0013 OPRINT3  D        U3
0014 OPRINT4  D        U4
0015 OPRINT5  D        U5
0016 OPRINT6  D        U6
0017 OPRINT7  D        U7
0018 OPRINT8  D        U8
```

Figure 15-22

Procedure PRINTS

```
RESPONSE MSG1404,#LIBRARY
// NOHALT 2,SYSTEM
// LOAD PRINTS
// PRINTER NAME-PRINT1,DEVICE-P1,FORMSNO-INVC
// PRINTER NAME-PRINT2,DEVICE-P2,FORMSNO-PAYR
// PRINTER NAME-PRINT3,DEVICE-P3,FORMSNO-BILL
// PRINTER NAME-PRINT4,DEVICE-P4,FORMSNO-STOK
// PRINTER NAME-PRINT5,DEVICE-P5,FORMSNO-QUOT
// SWITCH 11111000
// RUN
// NOHALT 0,SYSTEM
RESPONSE MSG1404N,#LIBRARY
```

Figure 15-23

Autoresponse source member MSG1404

```
SYS
1404 1,2 On printer xx change to forms number xxxx
```

Figure 15-24

Autoresponse source member MSG1404N

```
SYS
1404 N Return the response control back to the system operator
```

Automatically Responding to SYS-6300 Message

answered by Mike Patton

Q We always spool our printer output, but we don't always have the printer turned on. Consequently, some of our jobs halt and wait for an operator to respond to the system message SYS-6300, "Printer XX and the system are not communicating." Is there a way to respond automatically to this message?

A In the January 1986 version of the *S/36 System Messages* manual (SC21-7938-3), the message SYS-6300 has a severity level of 5, which indicates that no autoresponse is allowed. However, because you spool your output and because you know the cause of the problem, you can change the severity level of this message by creating the following source member (let's call it NO6300) and placing it in #LIBRARY:

```
SYS
```

```
6300 2,1
```

Member NO6300 operates on SYS (system) errors, specifically 6300, and specifies that option 2 should be taken automatically when the error occurs. In addition, member NO6300 specifies that the severity level of message SYS-6300 is to be treated as 1 rather than 5.

After you create member NO6300, you need to take two steps. First enter

```
RESPONSE NO6300,#LIBRARY
```

to effect the change to the message member. Then, enter

```
NOHALT 1,SYSTEM
```

to enable autoresponse for severity 1 (informational message) level errors.

If for some reason you decide to return message member NO6300 to its original state, you need to change the second line of member NO6300 to

```
6300 N,5
```

which will disable automatic response and reset the severity level to 5 as soon as you again run the RESPONSE procedure to update the message member.

Executing Spool Commands During High System Usage

answered by Mike Patton and Jeff Silden

Q We are having a problem with our spool file. We have a S/36 Model D2A with 2 MB of memory and 758 MB of disk. Our spool file is set at segments of 10 blocks and a size of 1,330 blocks. When we try to start, stop, or move spool file entries during our peak times, we receive system message 5852, "Unable to perform cmd now. Try again later." We had this problem before we upgraded to a Model D, and the upgrade has not alleviated the problem. I cannot find an explanation of a spool file interlock in any of my documentation. Please offer your comments.

A The situation described occurs when any "critical resource" is enqueued by one application without being dequeued in a reasonable amount of time. The spool file resource is considered one of those "critical system resources." System programs that require access to such a resource are programmed to "gracefully exit" using system error message 5852 so they won't otherwise have to wait on what might be a never-ending enqueue request. In your particular case, you could be keying the start, stop, or move commands at the precise moment when the system is also manipulating spool entries. Alternately, there may be programs installed and running on your system that manipulate the spool file (e.g., any of the printer passthrough products).

Another option is to reduce the spool file segment size. If a spool file segment is requested and, for whatever reason, is filled slowly, this might cause an interlock. Shops that print very large reports might set a large spool segment size, but for most shops, it can be set quite small.

Operation of the Spool File Interlock

answered by Mel Beckman, Mike Patton, and Jeffrey Pisarczyk

Q On occasion, I receive either system message SYS-4906 "Unable to perform OCL statement now" or system message SYS-5852 "Unable to perform command now" on my S/36.

The message occurs when I want to use the spool file and a lot of items are in it — it's not full, though. The SYS-4906 message is particularly bad because the procedure continues after the message is displayed. The message guide says that "the spool file interlock is not obtainable," which means nothing to me. What is the spool file interlock? And can I stop these messages from occurring?

A Space in the spool file is allocated in groups of sectors (called segments) instead of records (as in regular data files) and is assigned and released as reports are generated and printed or canceled. The spooling manager program, which is part of the SSP, maintains pointers of the "in use" and "available sector" chains. The logic for accessing/releasing disk space is tricky because multiple jobs could be concurrently allocating space. The spool file interlock allows multiple concurrent updates of the spool structure by serializing allocation and deallocation requests. Typically, if an area of a spool is busy, the SSP waits and retries after a couple of seconds. If the SSP must wait for an extended time, it sends the file interlock message, which means that another job is keeping the spool file busy and delaying other requests. File interlock messages usually occur when you write numerous new spool entries or when a third-party program improperly handles the spool interlock. Continuing entries aren't much of a load because the interlock is used only for new segments.

You shouldn't see a spool file interlock message regularly unless pointers within your spool file are damaged, your system is extremely overloaded, or you are using a third-party spool manipulation program that does not function properly (i.e., locks the spool file queue header for more than a short time). If none of these is the cause and you continue to receive the messages, report the problem to IBM.

Explanation of Spool File Size and Extents

answered by Mel Beckman and Mark Rubinstein

Q I just installed PTF 3700 and PTF 3704. Everything seems fine except for one problem: my spool file is configured for 4,500 blocks, but when I display the spool file status, it shows 27,000 blocks available, which is six times larger than configured. The catalog and the configuration each show 4,500 blocks, and the blocks available calculation works okay when I count down from 27,000. What's going on?

A The S/36 allows up to six extents of the spool file, which is why the print spool file is six times the number of blocks you specified in the configuration (i.e., 6 x 4,500 = 27,000). When the first extent fills up, another is created automatically on a different part of the disk; spool extents don't need to be contiguous. The D P command always shows you how many potential blocks you have, even if there isn't enough space for them all.

Printing on a Remote Printer

answered by Chuck Balsly and Bruce Hobbs

Q I want to print RPG reports and DW/36 documents as unattended remote operations. The remote location needs to be a standalone printer attached to an asynchronous modem with no PC or terminal attached. How can I do this?

A On the remote end, as long as the printer supplies a DTR (Data Terminal Ready) and uses X-on and X-off flow control protocol signals to prevent printer buffer overflow, a printer connected to a modem without a PC or terminal attached should work just fine. Find a protocol converter, such as PERLE GSD's model PDS350/294, that has autodial print control and initiates dial-out to remote printers under the control of a simple command line at the beginning of the spool file.

Transferring a Spool File Between a S/36 and an AS/400

answered by NEWS 3X/400 Staff

Q I have a S/36 that communicates with an AS/400. Is it possible to take the spool file from a S/36 and print it on an AS/400 without using Object Distribution Facility (ODF)?

A Yes. Use the COPYPRT command on the S/36 to copy the spool file to a data file, and send the data file to the AS/400. You do not need to write a program on the AS/400 to interpret the printer control codes in the COPYPRT file because the COPYPRT procedure and program in the AS/400

S/36 EE function just like their counterparts on the S/36. You can reprint a previously created COPYPRT file by using the following S/36 EE command:

```
COPYPRT NOCOPY,filename,,PRINT
```

The S/36 can produce only one type of spool file format when copying (using COPYPRT) to a data file. The AS/400, on the other hand, has multiple formats, including the same format as the S/36 in the CPYSPLF (Copy Spool File) command (using the keyword option CTLCHAR(*S36FMT)). This lets you move AS/400 printouts back to the S/36 for printing, if you want. Also, on the AS/400, if you use the parameter CTLCHAR(*FCFC) in the initial CPYSPLF command, you can copy the spool file data (that has been copied to a data file with the CPYSPLF command) back into an AS/400 printer output queue.

Programming with IPDS

by Michael Ingram

Code on diskette:
Procedure LTHD1$00
RPG program 1PDS0$02
Source members LETHDFIL, LOGO

Thousands of midrange system users are choosing Intelligent Printer Data Stream (IPDS) printers for their ability to produce and merge text and graphics in a variety of commercial applications. IPDS is also one of the cornerstones of IBM's SAA architecture and can function as a top layer printer protocol for BASIC, RPG, COBOL, Assembler, PL/I, and other SAA structured-programming languages. Ultimately, according to IBM, no other printer protocol will be supported in the SAA environment. Few users, however, are able to reap the full benefits of this versatile page-description protocol because application software that can exercise the full power of IPDS does not yet exist.

At present, unless you want to develop complicated custom programs, you must select from a limited number of commercial applications that use IPDS commands for producing and merging text and graphics. S/36 users, for example, can use the IPDS Advanced Functions PRPQ, IBM's IPDS interface software, to print bar codes and graphics with most IPDS printers. S/38 and AS/400 shops can use IBM's Graphical Data Display Manager (GDDM), Business Graphics Utility (BGU), or Presentation Graphics Routines (PGR) for IPDS graphics, charts, and text merge functions.

None of these applications, however, lets you go beyond the high-level commands they provide to take advantage of the IPDS printer's ability to provide local storage for reusable page elements. Using the printer's memory for storage of consistent elements eliminates the need for repeated downloading between pages or documents. For instance, with current software, you can't store forms overlays, scaled images, and signatures in an IPDS printer and merge these consistent elements with different fonts in a single document. Nor does current software let you tie together different applications such as RPG II text, DisplayWrite graphics, and GDDM pie charts.

To accomplish these tasks, you must exercise the printer's full IPDS capabilities by using data streams to send low-level (i.e., printer-specific, user-developed) commands. This article briefly examines the IPDS protocol and then describes how you can send a basic data stream using RPG II on the S/36 to create letterhead text and a logo.

IPDS Protocol

IPDS controls a printer on the basis of pages rather than paragraphs or lines. In addition to producing and merging text, images, graphics, and bar codes, IPDS manages downloaded resources (e.g., fonts, overlays, and page segments), controls device functions (e.g., duplexing, media bin selection, and output finishing), and handles exception functions (which include more than 200 possible errors ranging from invalid commands and data to invalid position on page).

By shifting much of the processing from the host to the printer, IPDS offers a less CPU-intensive means for producing high-volume graphics and bar codes than Advanced Printer Function (APF), BGU, and Magnum QMS boards.

IBM refers to IPDS printers as *state machines,* meaning that commands are defined within operating states that correspond to the element being printed. IBM recognizes nine IPDS states:

• Home: the initial operating state to which the printer returns at the end of each loaded page, page segment, coded font, or overlay

• Block: four states — IO Image, IM Image, Graphics, and Bar Code — in which the printer establishes the initial processing conditions for a block of data

• Page: the state that prints a logical page

• Overlay: the state that handles the storage of overlay data in the printer

• Page Segment: the state that allows storage of page segment data in the printer

• Any: a state for "Execute Order Any State" commands (e.g., exception handling control and print quality control) that can be received in any IPDS operating state

As the printer builds a page image in memory, it moves from state to state, storing graphics, fonts, and overlays until it receives a "page" command, which closes the page and returns the printer to home state. The host normally controls these states through a sequence of low-level command streams. However, to send the low-level commands necessary to merge text with graphics, create an overlay, or change a font, you must issue commands to force the printer to return to a specific state such as home.

IPDS Architecture

The IPDS architecture consists of a *device-control set* of control commands and eight functional areas, or *towers*, each containing a set of IPDS commands for a major printer capability. The device-control set encompasses all IPDS commands that set up a page, communicate device controls, and manage printer acknowledgment protocol. The eight towers include:

• Text: commands required to present text information on a page, a page segment, or an overlay

• IM Image: commands required to present raster image data (raster images are rectangular arrays of print data consisting of picture elements (PELS), where each PEL consists of one dot)

• IO Image: commands required to present additional raster image data functions, such as those controlling image compression and scaling

• Graphics: commands and orders (i.e., subcommands) required to present vector graphics

• Bar code: commands and data controls required to present machine-readable bar code

• Page Segments and Overlays (two towers): commands required to store and present constructs containing text, graphics, image, and bar-code information

• Loaded Font: commands required to load and delete font information

To claim IPDS support, IBM says a product must implement all commands in the device-control set (i.e., those concerned with error reporting and acknowledgment of commands), at least one subset of the eight towers, and all required commands, orders, and controls for each supported tower or subset. For more information about IPDS protocol and architecture, see Further Reading, page 500.

Communicating with an IPDS Printer

When you send a regular SNA Character String (SCS) job to an IPDS printer, the spool writer examines the device configuration and converts the SCS command format to an IPDS command format before sending it to the printer. Figure 15-25 summarizes the available methods and tools for accessing specific functions of IPDS from a S/36 host.

Figure 15-25
S/36 IPDS interfaces

Application	IPDS Options
1. Business Graphics Utilities	Create bar charts, pie charts, line charts, surface charts, scatter diagrams, Venn diagrams, text only diagrams, and histograms.
2. DisplayWrite/36 Graphics and Text merge	Merge text and graphics, such as charts and diagrams from BGU, on a single printed page using the INCLGRPH command.
3. High-level languages (HLLs)	Print typical text listings and applications from S/36 Assembler, BASIC, COBOL, and RPG II.
4. Intermixing IPDS data streams with HLLs	Send most IPDS commands to the printer using the IPDS Advanced Functions PRPQ transparent features.
5. IPDS Advanced Functions PRPQ	Print text, graphics, and bar codes selecting: LPI, font, text, bold, print quality, color, filled areas,graphics segments, character size, and character orientation. Draw circles, lines, and filled areas. Select filled area patterns, line type, line width, and page position. Print bar codes: MSI, UPCA, UPCE, UPC2, UPC5, EAN5, EAN8, EAN13, EAN2 add on, EAN5 add on, 2 of 5 Industrial, 2 of 5 Matrix, 2 of 5 Interleaved, and 3 of 9 codes.

For the S/36, the IPDS Advanced Functions PRPQ (PRPQ number P84094 for S/36 models 5360 or 5362 and PRPQ number P84095 for the 5363 or 5364) lets you select printer options such as CPI, LPI, font, and color and lets you print graphics and bar codes. In addition, you can use this PRPQ to intersperse printer command streams and data within a high-level language application. You must use the data stream if your printer supports a particular function (e.g., some text attributes or overlays) that is not available in the high-level language.

The IPDS command streams you send to an IPDS printer using the Advanced Functions PRPQ are embedded within a standard group of initialization commands from the host system. If you need to override any of the standard set of commands, you must send that particular command with the parameter(s) desired. You can send any command streams that are supported by your IPDS printer, except for those that request an acknowledgment (e.g., the Sense Type and Model command). In this case, the printer would return the information requested, but the application would not be able to handle the data, and you may encounter unpredictable results.

The IPDS Advanced Functions PRPQ does not include documentation to explain how to write and send IPDS commands to generate sophisticated commercial output, such as routines for in-house publishing or color printing or programs that let you vary elements of reports or forms by customer. To access these printer capabilities, you must use the IPDS transparent data stream function of the PRPQ to send command data streams that are similar to PC printer control codes used to change character density or font. These data streams consist of printer commands such as Write Text or Begin Page and data that is specific to each command.

IPDS Command Structure

You use hexadecimal notation to represent values in an IPDS low-level command. The structure of a typical IPDS command data stream is:

FORMAT	BYTE
LLLL	0-1 (byte count)
D6xx	2-3 (command)
FF	4 (Flag)
CCCC	5-6 (Correlation ID)
Data	7-n

Byte count refers to the two bytes that indicate the total number of bytes in the command stream. Next, hex D6 is followed by a command identifier (xx). The first bit of the flag byte indicates whether a response is required from the printer for the particular command. If the first flag bit is turned on, the printer must respond either positively or negatively. If the second bit is turned on, a correlation ID is present. The correlation ID is an optional two-byte identifier for identification of a particular command in a sequence of commands. Finally, the data can include parameters, subcommands, orders, data fields, and operands for specific commands.

To select near-letter-quality print (NLQ), for example, the IPDS command (without the optional correlation ID) is: 0008D63300F800FE. (Remember, all values placed in the command data stream are in hex notation). 0008 indicates the command string is eight bytes long; D633 identifies an "Execute Order Any State" command; the 00 flag means no acknowledgment is required from the printer and a correlation ID is not included; and F800FE is data consisting of the Print Quality Control (PQC) order and PQC order data to select NLQ.

Creating IPDS Letterhead

Now that you're equipped with some understanding of IPDS and its command data streams, let's look at how you actually send a data stream to create a letterhead that combines graphics with nonstandard text.

A typical letterhead consists of a solid or shaded logo and nonstandard text that often is printed in double-wide, bold, or italicized text. Although a S/36 host does not support italic and double-wide text attributes, I use the letterhead in Figure 15-26 to demonstrate how you can use printer data streams to select these attributes and also produce the arc segment of the filled area.

Letterhead Text

Figure 15-26
Sample letterhead

The text attributes of the letterhead require a LFE (Load Font Equivalence) command to the printer that specifies double-wide, italicized print and bolding (the equivalence or E portion of the LFE command). Data streams specify the text to be printed — Casey's Bakery Supplies, in this example — as well as the line positioning and spacing within the line of printed text. Other commands used to print the text of this letterhead are:

- Exception Handling Control: to control error reporting
- Set Media Size: to specify the page size
- Load Page Position: to position the current print location on the page
- Load Page Descriptor: to specify printing attributes of the page
- Print Quality Control: to specify the print quality of printed text (to print the Courier font (000B) specified in the LFE command, the print quality must be NLQ or DP Text)
- Begin Page: to switch from Home State to Page State
- Write Text: to send a Set Character Attributes subcommand to select the local font ID to be used when printing text

The *IPDS Reference* manual (S544-3417) — but not the *IPDS Advanced Functions PRPQ* manual — provides general information to determine appropriate commands. Because each printer is different, you should also refer to the *Printer Product and Programming Description Manual* for the specific product you are using.

Letterhead Graphics

For the graphics segment of the letterhead, I chose high-level commands from the Advanced Functions PRPQ to select the print position, to draw lines, and to define the graphics window on the page (Figure 15-27). The positioning is designed for 11-inch-by-14-inch paper.

The arc requires data streams that the S/36 host does not normally support. To make coding IPDS commands easier, I use a "script" contained in

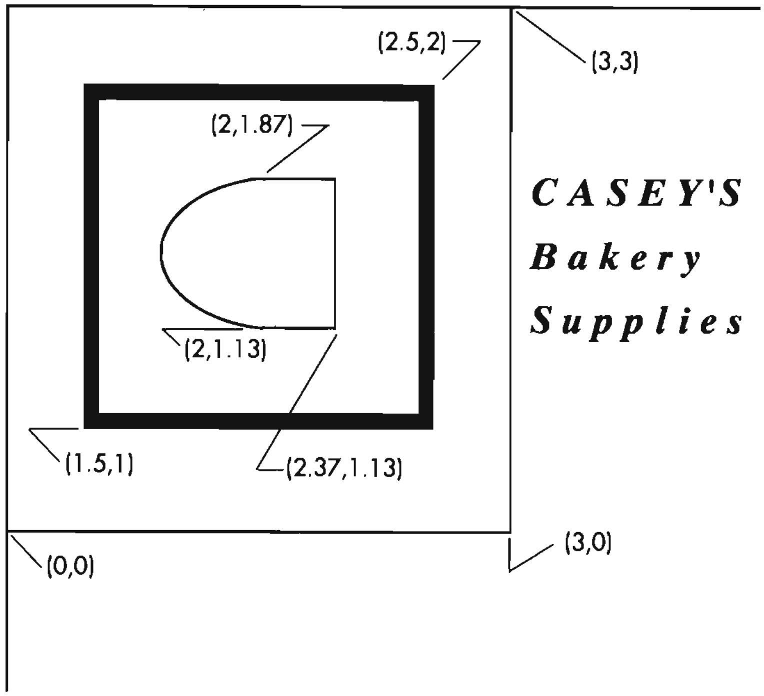


Figure 15-27

Positioning the logo in the graphics window

a source member to drive a general-purpose program that generates IPDS calls based on the scripted instructions. The source member in Figure 15-28 contains the PRPQ high-level commands and, because a high-level command was not available from the PRPQ, a graphics transparent command to print the arc using a data stream order (Fillet at Current Position, line 14). You format a source record as follows:

```
T FFFFFFFF00000000PPPP...PPP
```

The first position specifies the subroutine type (SUBTYP) and is either T, G, or B for text, graphics, or bar codes, respectively. These subroutines, provided by the IPDS Advanced Functions PRPQ, are available in RPG, COBOL, and Assembler versions. The graphics and bar code subroutines can be used only with IPDS printers.

Figure 15-28

Graphics source member LOGO

```
•       1         2         3         4         5         6         7         8
  G IPDSPRPOBEGSEG  0.00,0.00,3.00,3.00    Select graphics window
  G IPDSPRPOPOSITION1.50,2.00              Draw square
  G IPDSPRPOLINE    1.50,1.00
  G IPDSPRPOLINE    2.50,1.00
  G IPDSPRPOLINE    2.50,2.00
  G IPDSPRPQLINE    1 50,2.00
  G IPDSPRPQPOSITION2.00,1 13              Draw filled region 'C'
  G IPDSPRPQCOLOR   BLUE
  G IPDSPRPQPATTYPE 16
```

```
G IPDSPRPQBEGAREA
G IPDSPRPQLINE    2.37,1.13
G IPDSPRPQLINE    2.37,1.87
G IPDSPRPQLINE    2.00,1.87
G IPDSPRPQIGTRANS 8508065B08700B40065B
G IPDSPRPQENDAREA
G IPDSPRPQENDSEG                     End graphics window
```

Following SUBTYP is an eight-byte field (FF...) containing the printer output file name (in Figure 15-28, IPDSPRPQ). Positions 11 through 18 are an eight-byte option field (OO...) where you specify selected IPDS Advanced Functions PRPQ text, graphics, or bar-code options (for example, the LINE option in Figure 15-28). Parameters for the options are contained in an eighty-byte field (PP...) consisting of EBCDIC IPDS commands or orders.

Once the Casey's Bakery Supplies logo printed out as expected, I put the printer in buffer dump mode to record the actual commands sent to the printer to produce the graphics logo. (The printer buffer dump or packet dump mode of an IPDS printer is extremely useful in identifying what data streams the printer received and can help you determine what a typical sequence of commands is like.) The resulting commands from the S/36 host required to print the logo are a single Write Graphics Control command, two Write Graphics commands, and an End command. I merged these graphics commands with text commands as data streams to print the entire letterhead.

Printing It All Together

To merge the sample letterhead's text and graphics, I created S/36 procedure LTHD1$00 (Figure 15-29), RPG II program IPDS0$02 (Figure 15-30), and, using SEU, S/36 source member LETHDFIL (Figure 15-31) that contains IPDS command streams for use with any of the currently available IPDS midrange printers listed in Figure 15-32 or with any future IPDS-compatible printers.

Procedure LTHD1$00 uses the S/36 FROMLIBR procedure to copy the source member (LETHDFIL) to a sequential disk file (IPDSINP), to specify the output file (IPDSPRPQ), and to load and run the RPG program. In using procedure FROMLIBR, you must ensure that the record length (98) matches the size of the record specified for both the RPG input file and the source member that contains the data streams and that the name of the disk file (IPDSINP) matches the name or label specified as the input file in the RPG program.

RPG II Program IPDS0$02

Program IPDS0$02 reads disk file IPDSINP using a CHAIN command with a counter (X) to access records and processes them using S/36 RPG subroutines SUBR50 (Text or Printer Options), SUBR51 (Graphics), or SUBR52 (Bar Codes). (This example uses only SUBR50.)

You format the record the same way as the letterhead graphics source member. Following the 80-byte parameter field is a two-byte RTCODE field (returned by the subroutine) that specifies whether the subroutine was executed or an error occurred. The return codes — which are character, not hex — are listed in Figure 15-33.

You can include error-reporting or recovery in the RPG program based upon the return code. That is, if an error is returned by the return code, a message specific to the error can be posted. In the case of this program, any record encountered with an invalid field results in nothing being sent to the printer. Therefore, you can write comments throughout the input file and not affect the printed output.

The RPG program processes each record in the input disk file and checks the option field with each pass. When the option field read is equal to END, the RPG program turns on the last record indicator and terminates. The only output specification in the RPG program contains the printer output file name and ejects to page 1, line 1 before printing any data.

Source Member LETHDFIL

You format source member LETHDFIL exactly as previous records. The data in each line of the source member consists of the subroutine type, output file name, option, and parameter. The output file name must always match the output file name specified in the procedure and RPG program, in this case IPDSPRPQ. The option field is always IHTRANS (to send IPDS home state command data streams) except for the last record, which is set to END. The parameter field consists of the hex IPDS command streams; for example, parameter field 000AD63300F600E00002 (Figure 15-31, line 6) consists of the data stream for the Error Handling Control subcommand of the Execute Order Any State (XOA) command. (Refer to the IBM product-specific *Printer Product and Programming Description Manual* for information on each command.)

We have looked at a procedure and program that lets you combine graphics and nonstandard text using IPDS command streams. You could construct others that let you develop overlays, business reports, and form letters in which elements vary by customer or that use data streams for color printing, AIAG labels, or in-house publishing. Future articles will help you with these functions. Learning to communicate with your IPDS printer enables you to make better use of its local storage capacity and its power to incorporate print elements from different applications.

Further Reading

4224 Printer Product and Programming Description Manual (GC31-2551).

IPDS Handbook for the 3812 Printer (S544-3102).

Intelligent Printer Data Stream Advanced Functions PRPQ (GC21-9480).

Intelligent Printer Data Stream Reference (S544-3417).

Using the IBM Pageprinter 3812 with an IBM S/36 or S/38 (S544-3343).

Figure 15-29

Procedure LTHD1$00

```
***************************************************************************
****                                                                   ****
****    THIS IS A PROCEDURE TO CALL INDIVIDUAL RPG OBJECTS TO CREATE   ****
****    AN OUTPUT FILE FOR THE IPDS PRINTER  GRAPHICS - CREATE A       ****
****    LETTERHEAD FORM USING PRPQ COMMANDS AND DATA STREAMS.          ****
****                                                                   ****
****        1   A SOURCE MEMBER CONTAINING THE IPDS PARAMETERS FOR     ****
****            SUBROUTINE 50 & 51 (RPG GRAPHICS/TEXT ROUTINES).       ****
****            SEE SOURCE MEMBER.  LETHDFIL                           ****
****                                                                   ****
****        2   THIS PROCEDURE COPIES THE ABOVE SOURCE MEMBER          ****
****            (LETHDFIL) TEMPORARILY TO THE SYSTEM DISK.  THE        ****
****            TEMPORARY FILE IS USED TO PASS THE PARAMETERS TO       ****
****            THE SUBROUTINE USED, WITH AN RPG INPUT STATEMENT -     ****
****            SEE FILE NAMED:  IPDSINP                               ****
****                                                                   ****
****        3.  THE IPDS COMMANDS ARE MERGED WITH THE DATA OUTPUT IN   ****
****            AN OUTPUT FILE TO BE PRINTED.  THIS IS DONE WHEN THE   ****
****            RPG PROGRAM IS CALLED.  SEE RPG SOURCE MEMBER          ****
****            IPDSO$02                                               ****
****                                                                   ****
****        4.  THE ABOVE RPG PROGRAM WILL CREATE 1 OUTPUT FILE TO BE  ****
****            SPOOLED TO THE PRINTER  SEE OUTPUT FILE ON THE SPOOL   ****
****            WRITER  IPDSPRPQ                                       ****
****                                                                   ****
***************************************************************************
FROMLIBR LETHDFIL,SOURCE,IPDSINP,F1,3,175,  ,,98
// PRINTER NAME-IPDSPRPQ,PRIORITY-5
// FILE NAME-IPDSINP
// LOAD IPDSO$02
// RUN
```

Figure 15-30

Program IPDS0$02

```
*.        1         2    .    3  .  .. 4       .  5       .  6   .  .  7 ..  . 8
H         1                                                               IPDSO
F***********************************************************************
F* PROGRAM WHICH USES THE IPDS ADVANCED FUNCTIONS PRPQ.  THIS          *
F*        PROGRAM WILL ALLOW THE PROGRAMMER TO MERGE THE BAR CODES.    *
F*        GRAPHICS, AND TEXT OPTIONS OF THE PRPQ INTO A SINGLE         *
F*        PRINT JOB                                                    *
F*                                                                     *
F*        THE INPUT SOURCE FILE MUST SPECIFY THE SUBROUTINE USED       *
F*        (50, 51, OR 52), THE PRINTER OUTPUT FILE NAME, THE           *
F*        IPDS OPTION USED, AND THE ASSOCIATED PARAMETER VALUE.        *
F*                                                                     *
F*        THIS SOURCE CODE WILL CREATE A LETTERHEAD WITH A             *
F*        PREDEFINED LOGO.  THE LOGO CAN BE PRINTED ON THE PAPER       *
F*        TO BE USED TO PRINT ADDITIONAL TEXT  THIS EXAMPLE IS         *
F*        FOR EDUCATIONAL PURPOSES ONLY AND DOES NOT REFLECT A         *
F*        REAL APPLICATION.  THE IPDS DATA STREAMS ARE CREATED         *
F*        FROM THE SOURCE FILE  LETHDFIL                               *
F*                                                                     *
F*        WRITTEN BY   M  INGRAM                                       *
```

```
F*              DATE    12/02/88                                     *
F*           PROJECT    DDCC 6524 IPDS PRINTER                       *
F*********************************************************************
F****
F****  OUTPUT FILE FOR THIS PROGRAM IS IPDSPRPQ
F****
FIPDSPRPQO   F      132            PRINTER
F************************************************************************
F****  TEMPORARY DISK FILE (F1) USED FOR INPUT TO THIS PROGRAM IS
F****    IPDSINP  THE FILE CONTAINS THE SUBROUTINE USED (SUBTYP),
F****    THE FILE NAME (FNAME), THE OPTION USED (OPTION) AND THE
F****    PARAMETER VALUE (PARM) FOR IPDS ADVANCE FUNCTIONS PRPQ
F****
FIPDSINP IF  F      98             DISK
I************************************************************************
I****  INPUT A RECORD AND CHECK FOR A VALID FNAME, OPTION, AND PARM
I****    THE RETURN CODE (RTCODE) SPECIFIES WHETHER THE SUBROUTINE
I****    EXECUTES PROPERLY OR NOT
I****
IIPDSINP NS
I                                        1   1 SUBTYP
I                                        3  10 FNAME
I                                       11  18 OPTION
I                                       19  98 PARM
C           START     TAG
C                     Z-ADD1         X        30
C           LOOP      TAG
C           X         CHAINIPDSINP               91
C           X         ADD  1         X
C           SUBTYP    COMP 'T'                       20
C           SUBTYP    COMP 'G'                       21
C           SUBTYP    COMP 'B'                       22
C   20                EXIT SUBR50
C                     RLABL          FNAME   8
C                     RLABL          OPTION  8
C                     RLABL          PARM   80
C                     RLABL          RTCODE  2
C   21                EXIT SUBR51
C                     RLABL          FNAME   8
C                     RLABL          OPTION  8
C                     RLABL          PARM   80
C                     RLABL          RTCODE  2
C   22                EXIT SUBR52
C                     RLABL          FNAME   8
C                     RLABL          OPTION  8
C                     RLABL          PARM   80
C                     RLABL          RTCODE  2
C           OPTION    COMP 'END     '                92
C  N92                GOTO LOOP
C                     SETON                          LR
C           END       TAG
OIPDSPRPQD   01    1P
```

Figure 15-31

Source member LETHDFIL

```
FORMAT OF SOURCE MEMBER IS
     T FILENAMEOPTION__PARAMETER__________________________________________

     * ERROR HANDLING CONTROL
              TAKE ALTERNATE EXCEPTION ACTION
     T IPDSPRPQIHTRANS 000AD63300F600E00002

     * LOAD FONT EQUIVALENCES FOR TEXT STYLES *
              LOCAL FONT ID - 18
              DOUBLE WIDE, ITALICS, BOLD, 0 DEGREES, 10 CPI COURIER, US CHAR SET
     T IPDSPRPQIHTRANS 0015D63F0018001800000000002500080000008800

     * SET MEDIA SIZE *
              PAGE SIZE - 8 5 x 11 INCHES
     T IPDSPRPQIHTRANS 000ED6BF00170D00384O2FD03DE0
```

```
* LOAD PAGE POSITION *
        SET CURRENT PRINT POSITION
T IPDSPRPQIHTRANS 000FD66D000000004000000004000000

* LOAD PAGE DESCRIPTOR *
        USE LOCAL FONT ID - 1B
T IPDSPRPQIHTRANS 0030D6CF0000003840384000002FD000003DE00020000000000000000
T IPDSPRPQIHTRANS 00002D000000000000000FFFF0000FFFF1B0000

* XOA.  PRINT QUALITY CONTROL *                                        .
        TEXT QUALITY - NLQ
T IPDSPRPQIHTRANS 0008D63300F800FE

* BEGIN PAGE *
T IPDSPRPQIHTRANS 0009D6AF0000000000

* WRITE TEXT USING NEW LOCAL FONT *
        USE LOCAL FONT #1B
T IPDSPRPQIHTRANS 000AD62D002BD303F01B

* PRINT THE TEXT FOR THE RIGHT SIDE OF THE LOGO *
         SET CORRECT POSITION - BEGIN NEW LINES
T IPDSPRPQIHTRANS 001DD62D002BD302D82BD302D82BD302D82BD302D82BD302D82BD302D8
T IPDSPRPQIHTRANS 0009D62D002BD302D8
         'CASEY'S BAKERY SUPPLIES'
T IPDSPRPQIHTRANS 0022D62D004040404040404040
T IPDSPRPQIHTRANS 4040404040C3C1E2C5E87DE22BD302D82BD302D8
T IPDSPRPQIHTRANS 0022D62D004040404040404040
T IPDSPRPQIHTRANS 4040404040C281928599A8402BD302D82BD302D8
T IPDSPRPQIHTRANS 001BD62D004040404040404040
T IPDSPRPQIHTRANS 4040404040E2A497979389 85A2

* DRAW LOGO USING GRAPHICS DATA STREAMS
         WRITE GRAPHICS CONTROL
                GRAPHICS AREA POSITION CONTROL
T IPDSPRPQIHTRANS 003CD68400000BAC6B000000000000A000
                GRAPHIC OUTPUT CONTROL
T IPDSPRPQIHTRANS 10A66B00384010E010E0100000000000
                GRAPHIC DATA DESCRIPTOR
T IPDSPRPQIHTRANS 1CA6BB0000384038400000000000000010D410D40000000000000000
         WRITE GRAPHICS:
T IPDSPRPQIHTRANS 003DD68500700C000000000000002A000000000210408700B40
T IPDSPRPQIHTRANS 8104087005A081040E1005A081040E100B40810408700B
T IPDSPRPQIHTRANS 4021040B40065A2602000012810
         WRITE GRAPHICS
T IPDSPRPQIHTRANS 0033D68500700C00000000000060020000000000684081040D5206
T IPDSPRPQIHTRANS 5A81040D520A8281040B400A828508065B08700B40065B6000

         END GRAPHICS SEGMENT
T IPDSPRPQIHTRANS 0005D65D00

* SET HOME STATE *
T IPDSPRPQIHTRANS 0005D69700

* END JOB
T IPDSPRPQEND
```

Figure 15-32
Midrange printers that support IPDS

Vendor	Model	Type	Speed	Price
Decision Data Computer Corp.	6524-61	Matrix	400 CPS standard memory	$ 4,950
	6524-41	Matrix	400 CPS expanded memory	$ 5,300
IBM	3812-2	Electrophotographic	12 PPM	$ 9,490
	3816-2	Electrophotographic	24 PPM	$18,495
	4224-101	Matrix	200 CPS standard memory	$ 4,200
	4224-102	Matrix	400 CPS standard memory	$ 6,000
	4224-1E2	Matrix	400 CPS expanded memory	$ 6,500
	4224-1C2	Matrix	400 CPS color expanded memory	$ 6,700
Interface Systems Inc.	7224	Matrix	200/400 CPS	Not Available
Memorex-Telex	1224 (4 models)	Matrix	200/400 CPS	$ 4,200 to 6,700

Figure 15-33
Return codes

40 — normal completion
41 — the option field was invalid
42 — the parameter field was invalid
43 — an I/O error was detected
44 — the file name field was invalid

Programming

CHAPTER

16

Debugging RPG Program Dump Files

by Mel Beckman

Learn how to isolate bugs by producing and analyzing dump files.

Most S/36 programmers are familiar with IBM's stock-in-trade RPG debugging tool — the DEBUG statement. Most programmers also know how inadequate this tool is for real-world problem-solving because it displays only fields explicitly named on the DEBUG statement. Third-party interactive debugging aids improve the state of the art considerably, but these share a common fault with the DEBUG statement: you first must compile the program to run specifically in "debug" mode before doing any debugging, and then you must recompile it to take out the debugging when you are through.

Unfortunately, bugs don't always give you the kind of advance warning you need to isolate a failing program, edit it to insert debugging statements, recompile it, and try to re-create the original problem. Intermittent bugs are especially irksome: if you must leave debugging code in a suspect program until the bug reappears, you hinder performance and generate reams of unnecessary debugging output you'll later have to analyze. And if the problem is occurring at a remote site, your troubles become even more complicated. There must be a better way.

There is. The S/36, like most computers, is able to copy (i.e., dump) the contents of main storage to a file on disk for later analysis by a programmer. These files, called dump files, contain everything you need to track down many problems: the status of all indicators, the contents of variables, and the current instruction being executed. The symbol table in your RPG compile listing lets you determine the values of crucial variables and indicators from the dump file. While strictly a quick and dirty debugging tool, dumps are useful because they let you perform a thorough "postmortem" analysis of a failing program.

The dump facility is built into the S/36 and can be invoked at any time for any program — you don't need to add anything to your system to use it. And because dumps are stored in files on disk, they can be copied to diskette at a remote site and sent to your central programming site, giving you an important long-distance problem-solving tool. Learning how to produce and analyze dump files adds another weapon to your debugging arsenal and gives you a better idea of what's happening inside your machine.

To become a proficient dump debugger you must learn how to produce a dump file, how to use IBM's DUMP procedure to examine a dump, and how to use the dump information to isolate bugs in your programs.

Getting a Dump

There are three ways to get a main storage dump of your program:

1. Respond with option D to any system message that allows option 3.

2. Use the D option on a CANCEL command (e.g., CANCEL W1082345,D).

3. Run the IBM SETDUMP procedure.

The first two methods terminate your program after the dump is taken. This shouldn't be a problem because you're using dumps to debug particularly thorny problems that probably stop your program anyway. The third method, the SETDUMP procedure, lets your program continue execution after the dump, but using that method lies outside the scope of this article. Procedure SETDUMP is an advanced tool requiring more knowledge of RPG internals than this article covers.

An RPG array index error is a typical problem you might decide to debug with a dump. An error is generated when the array index value is negative, zero, or greater than the defined maximum. While RPG is quick to point out which array you've slighted, it doesn't tell you what index value provoked the error. A dump file reveals this secret to you. Figure 16-1 shows a small sample program, SAMPLE, that causes an array index error when it runs. The array index error causes the message:

```
RPG-9014 Options (0 23)
Index error in array DAY . . .
```

The message allows option 3, which means you can optionally request a dump by selecting option D. The D option results in a series of additional messages:

```
SYS-0016
Storage dump has been requested
SYS-1875
Task dump in progress to disk W1111858
SYS-1879 Options (01)
#DUMP.xx — Task dump taken to this file . . .
```

You should answer message SYS-1879 with option 0. Selecting option 1 prompts you for a diskette for saving the dump and failing program modules — something you would do normally only for remote sites that need to send the dump to you for analysis.

The system stores dumps on disk in files named #DUMP.*nn*, where *nn* is a two-digit number from 00 through 99. You can keep up to 100 such dump files on disk at one time (not a good idea if disk space is limited), and you can copy, rename, delete, save, or restore them just as you would any other disk file.

The IBM Dump Utility

After you obtain a dump file on disk, you can examine it using IBM's dump utility. (If your system has password security, to use the dump utility your user ID must have IBM Service Aid authority, which you can set using the SECEDIT procedure.) You invoke the dump utility using procedure DUMP:

```
DUMP MAIN,CRT,F1,#DUMP.nn
```

The first parameter, MAIN, tells the dump utility you want to examine a main storage dump file. The second parameter, CRT, indicates you want to browse the file interactively (you could optionally specify PRINTER to get a printed copy of the dump file). The third parameter is F1 for dump files on disk, or I1 for dump files on diskette (usually your dumps will be on disk). The last parameter is the name of the dump file you want to examine. If you omit this parameter, procedure DUMP displays information about the most recent dump file on disk and lets you browse among any other dump files on the same device (disk or diskette). If you want to browse the most recent dump file on disk interactively (the usual case), enter this abbeviated command:

```
DUMP ,CRT
```

Procedure DUMP then shows a status display for the dump file you selected (Figure 16-2).

The summary screen shows several important pieces of information that help you identify the dump you want to examine: the name of the dump file, the reason the dump was taken (usually "Storage dump has been requested"), the date and time the dump was taken, and the name of the procedure and program contained in the dump. (For dumps from remote sites, the SSP and microcode release levels might help you detect release compatibility problems.) Other values on this screen won't be used in simple RPG dump analysis. The bottom of the summary screen lists the command keys you can press for further action. If you didn't specify a particular dump file, you can use the Roll or Enter keys to page through all the dumps available. When you've determined that the dump file displayed is the one you want to examine, press Command key 1, and the contents of the dump file will appear (Figure 16-3).

The dump contents screen consists of three heading lines, containing much information you can ignore, followed by 256 bytes of dump data displayed in both hexadecimal and EBCDIC format. The rightmost column of the screen shows the EBCDIC translation of the 16 bytes on each line listed on the left side of the screen in hex. Each line of data is preceded by its beginning hex memory address. For example, the last line of data in Figure 16-3 begins at hex address 0000F0 in memory and contains the hex bytes D6C640C9C2D4 that correspond to the EBCDIC characters OF IBM. Again, the bottom line of the screen lists the available command keys. You can use the Roll or Enter keys to page through dump data 256 bytes at a time.

Procedure DUMP positions the cursor at the address field for the first data line, which is the only place you need to enter data for RPG debugging. The address field is six digits long, but the cursor is positioned at the third digit because the two leftmost digits usually stay set to 00. Immediately following the address field is a one-character storage option field that selects the kind of storage to be viewed. The only option you're interested in for RPG program debugging is X (for translated storage), although M (for real main storage) sometimes shows up on the initial display. (Procedure DUMP automatically sets this option for you as required, but if you key over it accidentally, reset it to X.)

Getting to the Top

The initial dump contents screen reflects the state of the job at the time the dump occurs. You'll probably need to look at an earlier state of the job because when the dump occurs, your job may actually be executing an SSP subprogram instead of your RPG program. An RPG program calls many SSP subprograms to perform tasks such as disk file operations, workstation input/output, and message display. In fact, when you obtain a dump by answering a system message with option D, your job is executing the system message subprogram when the dump occurs. Because each of these subprograms has its own 64 K region, or address space, the address information you use (from the symbol table in your RPG compile listing) to examine your RPG program is valid only when you're displaying data for your RPG program's address space. Furthermore, IBM subprograms themselves can call, or invoke, other IBM subprograms. Each such call increases the number of invocation levels through which you must backtrack to locate your RPG program's address space. Fortunately, procedure DUMP provides a command key that lets you quickly navigate to the top invocation level that contains your RPG program's region.

Each time you press Command key 5 (labeled scan at the bottom of the screen), procedure DUMP "backs up" one invocation level. When the invocation level changes, the numbers to the right of the RB and SB captions in the display heading change; if neither of these numbers changes after pressing Command key 5, you're at the top invocation level and the region for your RPG program is displayed. Thus, to display your RPG program region, simply keep pressing Command key 5 until neither the RB or SB numbers change. The address field (where the cursor is positioned) will read 000000X. Figure 16-4 shows the dump display of SAMPLE's RPG region.

Inside RPG

The Reserved Object Communication Area (ROCA) ia a data area reserved for the first 256 bytes of every RPG program. ROCA contains internal work areas and constants used by your program as well as a few other items of interest. Annotations on Figure 16-4 point out those values useful for debugging: the contents of the predefined field UDATE, the date and time the program was compiled, and the indicator array. You should match the compile date and time from the dump with that printed on the compile listing you use to debug the dump to verify that you're working with the right listing.

The indicator array contains one bit for every RPG indicator and several bits for RPG internal switches used for cycle control. You can decode this array using the table in Figure 16-5. Each line in the table represents one byte in the indicator array; the displacement into ROCA is the address of the byte. The indicators contained in a particular byte are listed on each line under the column heading for the hex value representing that bit. To decode the table values and find out which indicators were on when the dump occurred, translate the hex value to binary. Note which bits are on (i.e., have a value of 1). Next, match up the bit pattern with the table columns in Figure 16-5. For example, in the dump of ROCA for program SAMPLE (Figure 16-4), the indicator byte at hex address C3 is 60, which is 0110 0000 in binary. The second and third digits are 1, so you use the corresponding second and third columns, 40 and 20 respectively, in the indicator table in Figure 16-5. You can see the line for address C3 shows that indicator L0 is in the 40 column and that indicator LR is in the 20 column. Therefore, these indicators were on when the dump occurred.

The Heart of the Matter

Much of your sleuthing through a dump tracking down bugs consists of examining the values of variables that point a finger at your problem. For this task, you must refer to the symbol table portion of the RPG compile listing for your program, the section entitled EXECUTION TIME TABLES AND ARRAYS and FIELD NAMES USED. Figure 16-6 shows the symbol table for program SAMPLE.

The first section of the symbol table lists all the tables and arrays in your program along with their defined entry lengths and the number of entries. The column headed T/A DISP gives the hex address of the rightmost byte of the first element in each table or array. You can use this address to look directly at the contents of the array in the dump. In the example, the array DAY starts at hex address 010F. If the address you want to examine isn't currently on the screen, you can page to it, or you can enter it in the address field and press Enter. Procedure DUMP then displays the 256 bytes starting at the given address. Figure 16-7 shows the 256 bytes of program SAMPLE starting at hex address 0100, and it is annotated to show where the array DAY is located. Remember that the rightmost byte of the first element of array DAY is at 010F. The other elements follow contiguously (as usual in an array), as shown in the blocked-off hex and EBCDIC sections of the display.

The second section of the symbol table lists every field, its length, and its memory address. Refer again to Figure 16-6 and note that field X is located at address 0137, and field TODAY is located at address 0139. As with arrays, these addresses point to the rightmost byte of the field (except for data structure names, which point to the leftmost byte of the entire data structure). Figure 16-7 shows that the value of field X is 21 and the value of field TODAY is 38.

By looking at the value of X (which is used as an index to array DAY), you can see the problem with program SAMPLE. Field X is set to 21, but array DAY contains only 20 elements. Looking at the source listing for SAMPLE (Figure 16-1) reveals that the ADD statement in line 13 is the only statement where array DAY is referenced and indicator LR is on (as you determined by examining the indicator array). Line 13, therefore, is the cause of the array index error message. Further inspection reveals that the DO loop that automatically increments X from 1 through 20 exits with the value 21 in X — one higher than you expected. Now that you've isolated the bug, you can change the code to eliminate it.

Many intermittent bugs can be tracked down in just this manner — by looking at the state of variables and indicators to narrow down the range of suspect code.

Array Index Errors in Particular

Finding the array index error problem in program SAMPLE was easy because you had to look at only one index variable. But what if an array index error occurs for an array that is indexed with many different variables? Which of the indexes caused the problem?

A characteristic of RPG internals can answer this question quickly. When an array index error occurs, the contents of the Instruction Address Register (IAR) point very near to the address of the offending variable index. The value of the IAR appears to the right of the caption IAR in the screen heading; it is 8002C2 in the dump for program SAMPLE (Figure 16-7). For RPG dump debugging, you can disregard the 80 in 8002C2. The address you want is 02C2; type it in the address field, press Enter, and the screen shown in Figure 16-8 appears. The third and fourth bytes on the first data line *may be* the address of the variable index causing the problem — in this case 0137 — but you won't know until you check the symbol table. Referring to the symbol table for program SAMPLE (Figure 16-6), you can see that 0137 is the address of X, which is indeed the errant index. If 0137 were not found in the symbol table, by definition the error occurred on a MOVEA operation (a conclusion based on knowledge of how the RPG internal routine for the MOVEA operation sets up array indexes), and the address of the variable index is in the 10th and 11th bytes.

Even in a large program, once you've identified the variable index causing an array index error, it's easy to isolate which occurrence of that variable index is the culprit by checking the contents of nearby variables. Because a dump reveals the value of all variables and indicators, this task is straightforward in contrast to the old approach where you insert DEBUG statements everywhere the variable index is used, recompile the program, and go through possibly complicated maneuvers to reproduce the problem.

Getting in Deeper

You can extract even more information from a dump when armed with additional information about the internal structure of an RPG program.

One source for this — still available from IBM — is the *System/34 RPG Logic Manual* (LY21-0565). Although this volume ostensibly covers only the S/34, the internals of an RPG program on a S/36 are nearly identical. For more details on the dump utility and on procedure SETDUMP (which lets you dump a running job without terminating it), look in the IBM *System/36 Program Problem Diagnosis and Diagnostic Aids* (SY21-0593-5). This publication also discusses the IBM Dump File Analysis utility (a standard component of SSP), which you can use to get an overview of all jobs running on the system at the time a dump was taken — especially useful for analyzing dumps from remote sites.

While dumps aren't a cure-all for your debugging woes, they have their place on the S/36. When you must track down intermittent errors after they appear, knowing the basics of dump file analysis can keep you out of the dumps.

Figure 16-1

Program SAMPLE

```
*        1         2         3         4         5         6         7         8
0001 H                                                                     SAMPLE
0002 H*
0003 H* Sample program for RPG dump debugging
0004 H*
0005 E                   DAY        20  2 0
0006 C                    DO   20       X       20
0007 C                   MOVE UDAY      TODAY   20
0008 C                   ADD  X         TODAY
0009 C                   MOVE TODAY     DAY,X
0010 C                   END
0011 C*
0012 C                    SETON                     LR
0013 C                    ADD  30       DAY,X
```

Figure 16-2

Dump file summary display

```
                              S/36 MAIN STORAGE DUMP                           W1

   #DUMP.03      Task dump file                                               1 OF 4
   Storage dump has been requested
   Date  88/04/18   Time  11.28.27
   SSP      Rel 05   Mod 01
   MCODE    Rel 05   Mod 10
   TB    02C560    JCB  008D50    RB   02BF20    MIC  0016
   XR1   02BF60    XR2  02BF70    IAR  8011E9    ARR  00F4
   WR4   03D0      WR5  0000      WR6  00E0      WR7  8001
   PMSR  0A04      OP   F40104    ACE  000000    DIR    80
   Procedure    LIBRX             Program     SAMPLE




   Cmd1-Select this dump file        Cmd4-Delete this dump file          Enter-Page
                                     Cmd7-End                            Roll -Page
```

Figure 16-3

Dump contents screen

```
#DUMP.03                        S/36 MAIN STORAGE DUMP                                  W1
TB 02C560  RB 02BF20    IAR 8011E9  XR1 02BF60  WR4 03D0  WR5 0000  PMSR 0A04
           SB 000000    ARR 00F4    XR2 02BF70  WR6 00E0  WR7 8001  DIR  80

ADDR       00         04         08         0C
000000M    00000000   00000000   00000000   00000000     *                *
000010     00000000   00000000   00000000   00000000     *                *
000020     00019E00   00000000   00000000   00000000     *                *
000030     00000000   00000000   00000000   00000000     *                *
000040     00000000   00000000   00000000   00000000     *                *
000050     00000000   00000000   00000000   00000000     *                *
000060     00000000   00000000   00000000   00000000     *                *
000070     00000000   00000000   00000000   00000000     *                *
000080     00000000   00000000   00000000   00000000     *                *
000090     00000000   00000000   00000000   00000000     *                *
0000A0     F5F7F2F7   60E2E2F1   404DC35D   40C3D6D7     *5727-SS1 (C) COP*
0000B0     E8D9C9C7   C8E340C9   C2D440C3   D6D9D740     *YRIGHT IBM CORP *
0000C0     F1F9F8F3   6B40F1F9   F8F640D3   C9C3C5D5     *1983, 1986 LICEN*
0000D0     E2C5C440   D4C1E3C5   D9C9C1D3   406040D7     *SED MATERIAL - P*
0000E0     D9D6C7D9   C1D440D7   D9D6D7C5   D9E3E840     *ROGRAM PROPERTY *
0000F0     D6C640C9   C2D44040   40404040   40404040     *OF IBM          *

Cmd1-Restart      Cmd3-Dump file status      Cmd5-Scan     Cmd7-End     Roll-Page
```

Figure 16-4

Dump of RPG reserved object communication area

```
#DUMP.03                        S/36 MAIN STORAGE DUMP

TB 02C560  RB 01AA00   IAR 8002C2  XR1 800000  WR4 03D0  WR5 0000  PMSR 1F02

           SB 014240   ARR 00F4    XR2 8000B4  WR6 00E0  WR7 8001  DIR  80


ADDR       00         04         08         0C

000000X    C2F0F0F0   F0F0F0F0   F0F0F0F0   F0F0F0F0     *B000000000000000*

000010     F10D0000   00008008   00000500   0D300000     *1..............*

000020     00000000   00000017   00000000   00000000     *................*

000030     00000000   00008000   24808007   0004005E     *...............;*

000040     38010080   00008000   0000001C   00000000     *................*

000050     80080000   05001C30   00000000   00000000     *................*

000060     00170000   00000000   00000000   00000000     *................*

000070     80002480   8008003E   7B7BD4E2   C7D9D700     *.....##MSGRP....*

000080     00         00000000   00000000                *................*

000090     40FFFF00   00010002   00000000   00000000     * ...............*

0000A0     0000F0F4   F1F8F8F8   00001127   C08702D5           88....{g.N*

0000B0     C08701AE   0000C3D9   50880418   40400000     *{g...CR*h..  ...*

0000C0     00600060   00010000   00000000   00000000     *.-.-............*

0000D0     00000000   00000000   00800000   F5000000     *...........5....*

0000E0     00000000   00000000   00000000   00000000     *................*

0000F0     000000C4   C1E84040   40010221   010F005E     *...DAY  ......;*


Cmd1-Restart     Cmd3-Dump file status     Cmd5-Scan     Cmd7-End     Roll-Page
```

UDATE

Time and date of compile

Indicators

Figure 16-5

RPG II indicator table

Displacement into ROCA	Hex Byte Mask 80	40	20	10	08	04	02	01
C2	H4	H3	H2	H1	—	Mr (Int.)	MR (Ex.)	1P
C3	L1	L0	LR	H9	H8	H7	H6	H5
C4	L9	L8	L7	L6	L5	L4	L3	L2
C5	U1	U2	U3	U4	U5	U6	U7	U8
C6	KH	KG	KF	KE	KD	KC	KB	KA
C7	KQ	KP	KN	KM	KL	KK	KJ	KI
C8	KY	KX	KW	KV	KU	KT	KS	KR
C9	07	06	05	04	03	02	01	
CA	15	14	13	12	11	10	09	08
CB	23	22	21	20	19	18	17	16
CC	31	30	29	28	27	26	25	24
CD	39	38	37	36	35	34	33	32
CE	47	46	45	44	43	42	41	40
CF	55	54	53	52	51	50	49	48
D0	63	62	61	60	59	58	57	56
D1	71	70	69	68	67	66	65	64
D2	79	78	77	76	75	74	73	72
D3	87	86	85	84	83	82	81	80
D4	95	94	93	92	91	90	89	88
D5	—	—	—	—	99	98	97	96
D6	OV Ex.	OG Ex.	OF Ex.	OE Ex.	OD Ex.	OC Ex.	OB Ex.	OA Ex.
D7	OV 1st Int.	OG 1st Int.	OF 1st Int.	OE 1st Int.	OD 1st Int.	OC 1st Int.	OB 1st Int.	OA 1st Int.
D8	OV 2nd Int.	OG 2nd Int.	OF 2nd Int.	OE 2nd Int.	OD 2nd Int.	OC 2nd Int.	OB 2nd Int.	OA 2nd Int.
D9	Total cycle switch	Control fields processed	Overflow being processed	EOF on look-ahead	Close has been entered	**RESERVED		

Note: For each overflow indicator there are two internal indicators. The first internal indicator indicates that overflow has occurred; the second indicator indicates that the overflow output code has been fetched.

Ex. = External   Int. = Internal

Source: IBM

Figure 16-6

RPG compiler symbol table listing

```
EXECUTION TIME TABLES AND ARRAYS

STMT#    TABLE/  DEC  ENTRY  NUMBER OF   DTT     T/A
DEFINED  ARRAY   POS  LENGTH ENTRIES     DISP    DISP
 0001    DAY      0     002   00020      0100    010F

 FIELD NAMES USED
STMT#  NAME   DEC  LNG   DISP
 0003 UDAY     0   0002   00A5
 0002 X        0   0002   0137
 0003 TODAY    0   0002   0139
```

Figure 16-7

RPG region starting at hexadecimal address 0100

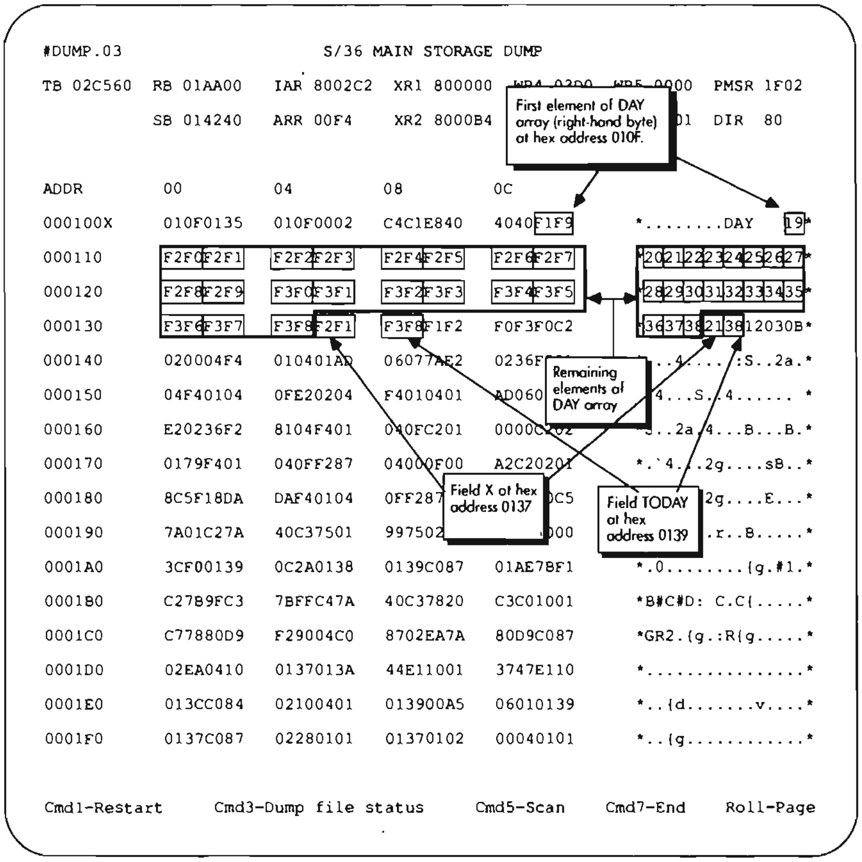


Figure 16-8

Locating the errant array index variable

1. Key value of the IAR into the address field.

2. See if the address in 3rd & 4th bytes is in the symbol table.

3. If not, the error was on a MOVEA, and the symbol address is in the 10th & 11th bytes.

```
#DUMP.00                      /36 MAIN STORAGE DUMP                              W1
TB 02E9C0  RB 00C840    IAR 8002C2   XR1 800000  WR4 5538  WR5 0000  PMSR 1F02
           SB 00CFA0    ARR 00F4     XR2 8000B4  WR6 00E0  WR7 8001  DIR  80

ADDR       00                      8
0002C2X    14100137    951C0102    21FD7502    FF351002      *....n...........*
0002D2     390007C2    0202E0F4    01040FF2    87048004      *...B..\4...2g...*
0002E2     00C5F401    04040000    7820C3C0    1002D57B      *.E4......C{..N#.*
0002F2     FFD77BFF    D87A02C2    7804C2F2    10037B02      *P#Q:.B.B2..#....*
000302     C2C08701    D2FFC1D5    C3C540D4    C1C9D5E3      *B{g.K.ANCE MAINT*
000312     C5D5C1D5    C3C540D4    C1C9D5E3    C5D5C1D5      *ENANCE MAINTENAN*
000322     C3C540D4    C1C9D5E3    C5D5C1D5    C3C540D4      *CE MAINTENANCE M*
000332     C1C9D5E3    C5D5C1D5    C3C540D4    C1C9D5E3      *AINTENANCE MAINT*
000342     C5D5C1D5    C3C540D4    C1C9D5E3    C5D5C1D5      *ENANCE MAINTENAN*
000352     C3C540D4    C1C9D5E3    C5D5C1D5    C3C540D4      *CE MAINTENANCE M*
000362     C1C9D5E3    C5D5C1D5    C3C540D4    C1C9D5E3      *AINTENANCE MAINT*
000372     C5D5C1D5    C3C540D4    C1C9D5E3    C5D5C1D5      *ENANCE MAINTENAN*
000382     C3C540D4    C1C9D5E3    C5D5C1D5    C3C540D4      *CE MAINTENANCE M*
000392     C1C9D5E3    C5D5C1D5    C3C540D4    C1C9D5E3      *AINTENANCE MAINT*
0003A2     C5D5C1D5    C3C540D4    C1C9D5E3    C5D5C1D5      *ENANCE MAINTENAN*
0003B2     C3C540D4    C1C9D5E3    C5D5C1D5    C3C540D4      *CE MAINTENANCE M*

Cmd1-Restart     Cmd3-Dump file status     Cmd5-Scan    Cmd7-End    Roll-Page
```

Debugging RPG Programs Using Conditional DEBUG

by Robert Griffiths

RPG DEBUG for the S/36 is certainly a useful, if verbose, tool. The function prints a status report for every program cycle, producing a potentially lengthy printout for long programs. But to find a bug, you probably don't need to examine every program cycle. The following procedure lets you avoid this information overload by making Command key 21 a toggle to turn the debug capability on or off. For example, you may want to turn the debug capability off during execution of those parts of a program you do not need to examine for bugs. Simply add the following lines to the procedure that calls the debug program:

```
// * 'Debug? (Y/N-default N)'
// IF ?R?=Y SWITCH XXXXXXX1
// ELSE     SWITCH XXXXXXX0
```

To make the procedure work in your program, you need to code a 1 in column 15 of the H-spec and include an F-spec for the debug print file. Then place the following line in the C-specs:

```
NU8            SETON                     88   +
 KV            ADD  5        DEBUG   10  88
```

Debugging RPG Programs Using DEBUG Files

by John E. King

If you're a real programmer, you probably wouldn't admit to using the DEBUG facility. But perhaps "somebody you know" finds it quite useful for tracking developmental problems. Perhaps this "somebody" would find it handier to route the DEBUG output to disk, where it could be displayed on a screen, than to wait for DEBUG output to print. The following steps route the DEBUG output to disk:

1. Place a 1 in column 15 of the H-spec to enable DEBUG. (After the program is working, simply remove the 1 to disable DEBUG.)

2. Include an F-spec for the DEBUG file, specifying an O in column 15, a record length of 132 bytes in columns 24 through 27, DISK in columns 40 through 43, and an A in column 66.

3. Use the BLDFILE procedure to create the DEBUG output file.

4. Code a // FILE statement in the procedure to reference the DEBUG file. (Use DISP-OLD to overwrite an existing file.)

5. Compile and execute the problem program.

At any appropriate point, you may interrupt the program. Use LIST-DATA (or POP's browser) to view the contents of the DEBUG disk file. This technique is particularly useful when developing interactive programs because it gives you the option to view immediately the result of each cycle through the program.

Profiling an RPG Program

by Mel Beckman

Code on diskette:
Procedures PROFRPG, PROFPRT
RPG programs PROFL1, PROFL2, PROFL3

What would you say if I offered you a S/36 programming tool that could show you where RPG program tune-ups are needed and how thoroughly programs have been tested? Well, such a tool, called a program profiler, is available and can be yours for a small investment of three to four hours of your time. A profiler can save you hundreds of programming hours that you might otherwise spend tuning your programs or chasing bugs that should have been caught during testing. Written entirely in RPG and requiring no assembler code or patches, a profiler can be used on any S/36 with an RPG compiler. With only a few minor modifications, you can migrate a profiler to the AS/400. And even if you're a non-RPG shop, you can use this profiler as a model to build a profiler for your language of choice. So read on for the inside scoop on the science of profiling, how to build a profiler, and how to put the profiler to work for you.

Profile of a Profiler

Profilers are a stock-in-trade programming tool; several different profiling techniques have evolved over time. Some profilers require special hardware and super-accurate timers. Some use interrupts at random intervals to inspect the programs and build statistical maps that show where interrupts occur most frequently. Each profiling technique has advantages and disadvantages. Hardware profilers yield high accuracy, but require expensive equipment. Statistical profilers, which require no special hardware, produce accurate results over long runs, but are inaccurate for short execution times or for interactive transactions.

The profiler I describe here uses a technique called statement counting, which is the easiest profiling technique to implement and provides accurate reporting regardless of the execution time. The profiler works by counting the number of times each program source statement is executed. An example of a source listing produced by the profiler appears in Figure 16-9. Only C-specs are executed in RPG, so the profiler prints only the calculation part of the code. The leftmost number on each statement shows the number of times the statement is executed during a test run. Statements executed most frequently are probably the ones consuming the most time, while statements that aren't executed at all during a test program run aren't tested and therefore flag inadequacies in your test data.

The profiled program used in this example is program FSMOCL. I produced the sample profile shown in Figure 16-9 by running program FSMOCL against a batch of test data that attempts to exercise every part of the code.

Inspect Figure 16-9, and you can see that the parts of the program executed most frequently are the "interpreter loop" (lines 47 through 54), the GETSYM routine (lines 62 through 70), and the DO routine (lines 76 through 83). The next busiest part of the program is the SC routine (lines 87 through 92). Clearly, any optimizing that reduces the number of times these statements are executed has the best chance of speeding up the program.

Some RPG statements, such as comments and END statements, are not actually executed and thus have no statement counts. However, one statement in program FSMOCL that should be counted, but isn't, is the MOVE instruction on line 68, which, according to the comments, is executed when the program senses an "end-of-line" condition. That this line is never executed indicates that the test data is incomplete — it never includes an end-of-line case. This example illustrates well the profiler's value in measuring test coverage. Without the profiler, this test data flaw might not be discovered until the program fails in a production environment, which is the very disaster you try to avoid by testing programs!

Running the RPG Profiler

The RPG profiler consists of two procedures, PROFRPG and PROFPRT, and three programs, PROFL1, PROFL2, and PROFL3. You use the profiler in a three-step process. In the first step, you run procedure PROFRPG:

PROFRPG *program,library*

where *program* is the name of the source program being profiled, and *library* is the name of the library containing the program. This step inserts extra RPG statements — called instrumentation code — into the RPG program being tested, creating a new version of the source program.

In step two, you compile the newly "instrumented" source program and then run the resulting object program in its normal environment. During execution, the instrumentation code collects statement execution counts that are written into a data file when the program ends. In step three, you run procedure PROFPRT to print the profiled source listing:

PROFPRT *program,library*

where once again, *program* is the name of the profiled program, and *library* is the name of the library containing the program.

Profiling Mechanics

Before you can examine the profiler's procedures and programs in detail, you must understand how the profiler inserts instrumentation code into a profiled program. Program SAMP (Figure 16-10a) shows the RPG source code for a simple program without instrumentation statements. Figure 16-10b shows program SAMP after the profiler adds the instrumentation statements. For identification when printing the profile report, the profiler marks the added statements with #+ in positions 4 and 5.

The first instrumentation line is an F-spec instructing the file to contain statement execution counts. This file's name is P#*pppppp*, where *pppppp* is the program name. The next instrumentation line is an E-spec that defines an array of statement execution counts. (The name of the array is #; the name must be only one character long so it can fit in the result field of subsequent ADD instructions with four-character index values.) This array contains one six-digit element for each statement being counted. Notice that the file name from the F-spec also appears in positions 19 through 26 of the E-spec. This combination of F-spec and E-spec uses a convenient feature of RPG called *end-of-job array output*, which automatically writes the contents of the profile array (#) at the end of the job to file P#SAMP without requiring O-specs for file P#SAMP.

The remaining instrumentation statements consist of one ADD instruction for every counted statement. Each ADD instruction increments a particular element in array #. The profiler places each ADD instruction so it is executed whenever its associated source statement is executed. Some source statements, such as comments and END (among others), are never executed and therefore aren't counted with instrumentation ADD statements. The profiler also uses positions 93 through 96 of the original source statement to store the corresponding array element number, which is used to retrieve the correct execution count for each original source statement when the profile listing is printed.

You need not worry about the extra time and memory (i.e., overhead) used by the instrumentation code, except in programs approaching a 64 K compiled size (obtained from the end of the compilation listing). Execution overhead isn't important because the profiler's results are unaffected by the time consumed by instrumentation code, and you don't leave instrumentation code in production programs. Memory overhead is 12 bytes per counted instruction — six for the array element, and six for the single machine instruction generated by an ADD to a literal array element. The P#*pppppp* output file requires a fixed overhead of about 1 K. A program containing 1,000 executable C-specs increases in size by only 13 K.

How It Works

Profiling requires three steps: inserting instrumentation code, compiling and running the profiled program, and printing the profiled source listing. Procedure PROFRPG (Figure 16-11) carries out the first step of profiling — inserting instrumentation code — by calling two RPG programs. Program PROFL1 (Figure 16-12) reads the original source program and inserts most of the instrumentation code. Because program PROFL1 makes only one pass over the original source code and doesn't know how many statement counters it needs until the end of that pass, it can't specify the number of elements in the # array E-spec when it inserts that statement. So, in the LDA, program PROFL1 stores the number of elements needed, and Program PROFL2 (Figure 16-13) retrieves that value and inserts it into the E-spec. The last step in procedure PROFRPG copies the instrumented source member back into the source library under a new name with the form P$*pppppp*, where *pppppp* is the original program name. It is this new source member you must compile to get an instrumented object program.

Program PROFL1 uses an unusual technique to read the original source program. Procedure PROFRPG (Figure 16-11) contains a // COM PILE statement between the // LOAD PROFL1 and // FILE NAME-$SOURCE statements. The program uses the // COMPILE statement to automatically copy a specified library source member into a job file named

$SOURCE. You indicate the member name and library being copied with the SOURCE and INLIB parameters of the // COMPILE statement. This technique eliminates the extra step of calling $MAINT to copy a source member to a disk file for processing by an RPG program.

The $SOURCE file contains 96-byte records. Unlike a $MAINT-generated file, however, the $SOURCE files do not contain // COPY and // CEND statements to delimit the source member. Instead, $SOURCE marks the end of the file with a record containing /* in positions 1 and 2. You must set program PROFL1's Source Required attribute in the compiled object member for the // COMPILE statement to work. You do this by compiling the program with the OCL statements shown in Figure 16-14. The RPGC procedure's NOLINK and OBJECT parameters generate an intermediate R-module for program PROFL1 instead of an automatically "linked" O-module from RPG. The OLINK (overlay linkage editor) procedure then performs the link edit step to produce an O-module (line 2 of Figure 16-14). The SRQ parameter of the OLINK procedure sets the Source Required attribute in the resulting O-module, thus enabling the // COMPILE statement.

You must carry out the second profiling step — compiling and running the profiled program — manually. The profiler can't compile the target program automatically because the program may require special values (such as MRTMAX) on the RPGC compile procedure; only you know the OCL and execution environment your program requires. When you compile the instrumented source program, remember that although the source member name (P$*pppppp*) differs from your original program name, the object program name is the same, and the instrumented object program replaces any existing version in the target library. To run your instrumented program, you must insert a // FILE statement for the P#*pppppp* file (which contains statement counts). Figure 16-15 shows the // FILE statement added to the sample program's OCL. You can leave this statement in your OCL even after removing instrumentation because SSP ignores it if your program doesn't actually open the file.

Procedure PROFPRT (Figure 16-16) carries out the last profiling step — printing the profiled source listing. It calls program PROFL3 (Figure 16-17) to merge the statement counts from the P#*pppppp* data file with the instrumented source code contained in member P$*pppppp*. Program PROFL3 uses the // COMPILE statement technique to read the instrumented source member, so be sure to compile program PROFL3 using the OCL in Figure 16-18 to set the Source Required attribute.

Count the Cost

Using a statement-counting profiler for performance tuning requires you to remember that all RPG statements aren't executed in the same amount of time. In particular, I/O operations take much longer than arithmetic operations. As a rule of thumb, you can use the table in Figure 16-19 to estimate the execution-time cost of various RPG statements. Arithmetic operations, such as ADD and SUB, and structural operations, such as COMP, DO, IF, and GOTO, are executed fastest because they run directly on the hardware in the S/36. Other arithmetic operations, such as MULT, DIV, and SQRT, are executed more slowly because the S/36 lacks multiply and divide hardware instructions, causing these operations to be carried out by subroutines. I/O operations are executed the most slowly because they must wait for the mechanical motion of devices such as disk arms and operator fingers. By using the factors in Figure 16-19, you can weight your profile statistics to give you a true measure of the time consumed by each statement.

Now Make the Profiler Work for You

You won't reap the benefits from useful tools such as this profiler unless you use them, so make this tool work for you by requiring its use in your shop. You should profile all of your regression (i.e., stored data) tests to ensure that the test data adequately exercises the code. You might even add a feature to the profile print program to flag executable lines that aren't executed. Also, before spending money on more memory or a faster CPU, profile your slowest applications to see if some simple coding changes won't ease your processing bottleneck. And keep a low profile.

Figure 16-9

Sample profiled source program listing

```
      *. . .  1 .   . . 2 . .  . 3 . .    4      . 5 . . . .  6 . . .  7    . 8
      0027 C*
      0028 C* Define local variables
      0029 C*
  365 0030 C                  MOVE *ZEROS  N       30         Next char index
  365 0031 C                  MOVE *ZEROS  S       30         Saved char index
  365 0032 C                  MOVE *ZEROS  STATE   30         Machine state
  365 0033 C                  MOVE *ZEROS  X       30         Column index
  365 0034 C                  MOVE *BLANKS SYM     6          Input symbol
  365 0035 C                  MOVE *BLANKS ACTION  2          Action code
  365 0036 C                  MOVE *BLANKS KIND    8          Kind of name
      0037 C*
      0038 C* Initialization
      0039 C*
  365 0040 C                  Z-ADD1       STATE              Set initial state
  365 0041 C                  EXSR RD                         Read a line
    1 0042 C   LR             Z-ADDO       STATE              Quit if no input
      0043 C*
      0044 C* Interpreter loop
      0045 C*
  365 0046 C         STATE    DOWNEO                          While STATE=0
27755 0047 C                  EXSR GETSYM                      Get next symbol
27755 0048 C                  Z-ADD1       X                   Initialize lookup
27755 0049 C         SYM      LOKUPCOL,X               11      Lookup column
15470 0050 C  N11             Z-ADD7       X                   (Default is other)
27755 0051 C                  MOVEASTT,STATE ROW,1             Extract table row
27755 0052 C                  MOVELROW,X   STATE               Set new state
27755 0053 C                  MOVE ROW,X   ACTION              Save action code
27755 0054 C                  EXSR DO                          Perform the action
      0055 C                  END                             End DO
      0056 C*
      0057 C* Routine to get the next symbol
      0058 C*
      0059 C*  Returns the next one-character symbol in the input line,
      0060 C*  or 'eol' if the end of the line is encountered
      0061 C*
27755 0062 C         GETSYM   BEGSR
27755 0063 C                  MOVE *BLANKS SYM                Clear symbol field
27755 0064 C                  ADD  1       N                  Bump to next chr
27755 0065 C         N        IFLE 120                        If not end-of-line
27755 0066 C                  MOVE INP,N   SYM                 Save the symbol
```

```
       C                      ELSE                                    Else it's e.o.l.
       C                      MOVE 'eol'     SYM                       So return 'eol'
       C                      END                                     End IF
 27755 C                      ENDSR
       C*
       C* Routine to do the specified action
       C*
       C*  Input. ACTION contains the action code to execute
       C*
 27755 C           DO         BEGSR                                   Depending on ACTION
       C           ACTION     CASEQ'sc'      SC                        Save character
       C           ACTION     CASEQ'pi'      PI                        Print identifier
       C           ACTION     CASEQ'pk'      PK                        Print keyword
       C           ACTION     CASEQ'pp'      PP                        Print parameter
       C           ACTION     CASEQ'rd'      RD                        Read next line
       C                      END                                     End CAS
 27755 C                      ENDSR
       C*
       C* Routine to save symbol
       C*
 16016 C           SC         BEGSR
 16016 C           S          IFLT 120                                If STR not full
 16016 C                      ADD  1         S                         Bump to next chr
 16016 C                      MOVE SYM       STR,S                     Save symbol
       C                      END                                     End IF
 16016 C                      ENDSR
       C*
       C* Routine to print identifier
       C*
   364 C           PI         BEGSR
   364 C                      MOVE '  Ident·'KIND                     Set kind
   364 C                      EXCPTOLINE                              Print kind and name
   364 C                      MOVE *ZEROS    S                        Reset STR index
   364 C                      MOVE *BLANKS   STR                      Clear STR array
   364 C                      ENDSR
       C*
       C* Routine to print keyword
       C*
  1183 C           PK         BEGSR
  1183 C                      MOVE 'Keyword:'KIND                     Set kind
  1183 C                      EXCPTOLINE                              Print kind and name
  1183 C                      MOVE *ZEROS    S                        Reset STR index
  1183 C                      MOVE *BLANKS   STR                      Clear STR array
  1183 C                      ENDSR
       C*
       C* Routine to print parameter
       C*
  1456 C           PP         BEGSR
  1456 C                      MOVE '  Param·'KIND                     Set kind
  1456 C                      EXCPTOLINE                              Print kind and name
  1456 C                      MOVE *ZEROS    S                        Reset STR index
  1456 C                      MOVE *BLANKS   STR                      Clear STR array
  1456 C                      ENDSR
       C*
       C* Routine to read next line
       C*
   729 C           RD         BEGSR
   729 C                      MOVE *ZEROS    N                        Reset INP index
   729 C                      MOVE *BLANKS   INP                      Clear INP array
   729 C                      READ INPUT                            LR Read next record
   728 C  NLR                 EXCPTILINE                              Print input line
   729 C                      ENDSR
```

Figure 16-10a

Program SAMP without instrumentation statements

```
*       1         2 . . . . 3     .   4         5         6         7   .     8
0001 H                                                                      SAMP
0002 F*
0003 F* A small sample program illustrating the use of the profiler
0004 F*
0005 FPRINT   O   F      132            PRINTER
0006 C*
0007 C* Compute the result of compound interest on $21 over 200 years at 5%
0008 C*
0009 C                         Z-ADD21        PRIN    92       Principle is $21
0010 C                         DO   200                        For 200 years
0011 C                         MULT 1.05      PRIN              Compound interest
0012 C                         END                             End DO
0013 C*
0014 C                         SETON                      LR
0015 OPRINT   T        LR
0016 O                                       24 'The result is'
0017 O                             PRIN  J   34
```

Figure 16-10b

Program SAMP after insertion of instrumentation statements

```
*       1 .       2         3       . 4 .       5         6         7         8
0001 H                                                                      SAMP
0002 F*
0003 F* A small sample program illustrating the use of the profiler
0004 F*
0005 FPRINT   O   F      132            PRINTER
   #+FP#SAMP  O          6              DISK
   #+E             P#SAMP  #       1    4  6 0
0006 C*
0007 C* Compute the result of compound interest on $21 over 200 years at 5%
0008 C*
   #+C                     ADD  1         #,0001
0009 C                     Z-ADD21        PRIN    92       Principle is $21
0001
   #+C                     ADD  1         #,0002
0010 C                     DO   200                        For 200 years
0002
   #+C                     ADD  1         #,0003
0011 C                     MULT 1.05      PRIN              Compound interest
0003
0012 C                     END                             End DO
0013 C*
   #+C                     ADD  1         #,0004
0014 C                     SETON                      LR
0004
0015 OPRINT   T        LR
0016 O                                       24 'The result is'
0017 O                             PRIN  J   34
**
```

Figure 16-11

Procedure PROFRPG

```
*
* Preprocess an RPG program to insert profiling code
*
* Parameter 1  Program name (six characters or less)
*           2. Source library name
*
// * 'PROFRPG procedure is running'
// LOCAL OFFSET-201,DATA-'?1?',BLANK-10
*
* Read the RPG source program and insert profiling code, producing file NEWSRC
*
// LOAD PROFL1
// COMPILE SOURCE-?1?,INLIB-?2?
// FILE NAME-$SOURCE,RETAIN-J,BLOCKS-50,EXTEND-100
// FILE NAME-SRCOUT,RETAIN-J,LABEL-NEWSRC,RECORDS-100,EXTEND-500
// RUN
*
* Plug number of elements into E-spec for '#' array
*
// LOAD PROFL2
// FILE NAME-SRCIN,LABEL-NEWSRC,RETAIN-J
```

```
// RUN
*
* Copy the instrumented source member from file NEWSRC back to the library
*
// LOAD $MAINT
// FILE NAME-NEWSRC,RETAIN-J
// RUN
// COPY FROM-DISK,FILE-NEWSRC,TO-?2?,RETAIN-R
// END
```

Figure 16-12

Program PROFL1

```
*.  . 1 ...  .. 2 .. .    3     .. 4     .    5 ..   . 6         7         8
     H     064                  B       1                             PROFL1
     F*
     F* RPG profiler phase 1: create new source member with profiling code
     F* Written by Mel Beckman
     F*
     F* NOTE: This program reads a library source member using the
     F*       // COMPILE OCL statement to create a $SOURCE file containing
     F*       the member  For this technique to work properly, you must
     F*       set the "source required" attribute in the compiled load
     F*       member for PROFL1  You do this by compiling and linking the
     F*       program in separate steps, using the following OCL.
     F*
     F*  RPGC PROFL1,library,,,,,,,,,,NOHALT,REPLACE,NOLINK,OBJECT (ten commas)
     F*  OLINK PROFL1,library,PROFL1,library,SRQ,,,#RPGLIB
     F*
     F$SOURCE ID  F9600  96            DISK
     FSRCOUT  O   F9600  96            DISK
     E*
     E* SRCIN text array
     E*
     E                    TXT        96  1
     I*
     I* Source input file
     I*
     I$SOURCE
     I                                        1  96 TXT
     I*
     I* LDA contains the program name end element count
     I*
     I           UDS
     I                                      201 206 PROGNM
     I                                      207 2100CNTR#
     C/EJECT
     C*
     C* Define internal variables
     C*
     C                    MOVE *BLANKS   ANOR    2
     C                    MOVE *BLANKS   CDONE   1
     C                    MOVE *BLANKS   CNTR#  40
     C                    MOVE *BLANKS   EDONE   1
     C                    MOVE *BLANKS   EOF     1
     C                    MOVE *BLANKS   FACT2   8
     C                    MOVE *BLANKS   FDONE   1
     C                    MOVE *BLANKS   HAVEC   1
     C                    MOVE *BLANKS   HAVEST  2
     C                    MOVE *BLANKS   INSB4   1
     C                    MOVE *BLANKS   ISTUFF 11
     C                    MOVE *BLANKS   OP2     2
     C                    MOVE *BLANKS   OP3     3
     C                    MOVE *BLANKS   OP5     5
     C                    MOVE *BLANKS   STAR2   2
     C                    MOVE *BLANKS   SLASHA  3
     C*
     C* Output the // COPY statement with a source member name of P$xxxxxx
     C* and read the first source statement
     C*
     C                    EXCPTPUTCPY                      Put // COPY
     C                    EXSR GETSRC                      Get 1st source stmt
     C*
     C* Flush source statements until F-spec for P# file is inserted
     C*
     C           FDONE    DOUEQ'Y'                         Until Fspec done
```

```
C*
C            TXT,6     COMP 'E'                      11  If we've
C   N11      TXT,6     COMP 'L'                      11   reached
C   N11      TXT,6     COMP 'T'                      11   the
C   N11      TXT,6     COMP 'I'                      11   Fspec
C   N11      TXT,6     COMP 'C'                      11   insertion
C   N11      TXT,6     COMP 'O'                      11   point
C    11                MOVE 'Y'       FDONE                Set flag
C*
C            FDONE     IFEQ 'Y'                          If Fspec goes here
C                      EXCPTFSPEC                         Insert it
C                      ELSE                              Else
C                      EXCPTPUTSRC                        Put source line
C                      EXSR GETSRC                        Get next source
C                      MOVE EOF       FDONE               Flag early EOF
C                      END                               End IF
C*
C                      END                              End DO
C/EJECT
C*
C* Flush source statements until E-spec for # array is inserted
C*
C            EDONE     DOUEQ'Y'                         Until Espec done
C*
C            TXT,6     COMP 'L'                      11  If we've
C   N11      TXT,6     COMP 'T'                      11   reached
C   N11      TXT,6     COMP 'I'                      11   the
C   N11      TXT,6     COMP 'C'                      11   Espec
C   N11      TXT,6     COMP 'O'                      11   insertion point
C    11                MOVE 'Y'       EDONE                Set flag
C*
C            EDONE     IFEQ 'Y'                          If Espec goes here
C                      EXCPTESPEC                         Insert it
C                      ELSE                              Else
C                      EXCPTPUTSRC                        Put source line
C                      EXSR GETSRC                        Get next source
C                      MOVE EOF       EDONE               Flag early EOF
C                      END                               End IF
C*
C                      END                              End DO
C*
C* Pass remaining statements, inserting counters as required
C*
C            CDONE     DOUEQ'Y'                         Until Cspec done
C                      EXSR QUALC                        Qualify the stmt
C*
C            HAVEC     IFEQ 'Y'                          If qualified Cspec
C                      EXSR INSRTC                        Insert counter
C                      ELSE                              Else
C                      EXCPTPUTSRC                        Put source line
C                      END                               End IF
C*
C                      EXSR GETSRC                       Get next source
C                      MOVE EOF       CDONE              Flag EOF
C*
C                      END                              End DO
C*
C* Emit ** and // CEND
C*
C                      MOVE *BLANKS   TXT
C                      MOVEA'**'      TXT,1
C                      EXCPTPUTSRC
C                      MOVEA'// CEND' TXT,1
C                      EXCPTPUTSRC
C*
C* End of job
C*
C                      SETON                     LR
C/EJECT
C*
C* Qualify a specification to see if it's a candidate for profiling
C*
C            QUALC     BEGSR
C*
C                      MOVE *BLANK    HAVEC             Clear flag
C                      MOVEATXT,1     STAR2             Extract **
```

```
C                      MOVEATXT,7     ANOR             Extract ANOR
C                      MOVEATXT,28    OP2              Extract part op
C                      MOVEATXT,28    OP3              Extract part op
C                      MOVEATXT,28    OP5              Extract whole op
C                      MOVEATXT,33    FACT2            Extract factor 2
C*
C         STAR2        IFEQ '**'                       If C.T  table hit
C                      MOVE 'Y'       HAVEST            Set flag
C                      END                             End IF
C*
C         TXT,6        COMP 'C'                    11   If C-spec
C   11    HAVEST       COMP 'Y'                1111      & not in tables
C   11    TXT,7        COMP '*'                1111      & not a comment
C   11    TXT,28       COMP ' '                1111      & not a no-op
C   11    ANOR         COMP 'AN'               1111      & not an AN
C   11    ANOR         COMP 'OR'               1111      & not an OR
C   11    OP5          COMP 'ELSE '            1111      & not an ELSE
C   11    OP5          COMP 'END  '            1111      & not an END
C   11    OP3          COMP 'CAS'              1111      & not a CASxx
C   11    OP3          COMP 'AND'              1111      & not an ANDxx
C   11    OP2          COMP 'OR'               1111      & not an ORxx
C   11    OP5          COMP 'RLABL'            1111      & not RLABL
C   11    OP5          COMP 'PARM '            1111      & not PARM
C   11    OP5          COMP 'PLIST'            1111      & not PLIST
C   11    OP5          COMP 'KLIST'            1111      & not KLIST
C   11    OP5          COMP 'KFLD'             1111      & not KFLD
C   11                 MOVE 'Y'       HAVEC               Set flag
C*
C                      ENDSR
C/EJECT
C*
C* Insert a C-spec counter statement, either before or after the original
C*
C         INSRTC       BEGSR
C*
C                      MOVE 'Y'       INSB4            Assume 'before'
C                      MOVEATXT,28    OP5              Extract whole op
C         OP5          COMP 'TAG  '                11  If TAG
C  N11    OP5          COMP 'BEGSR'                11   or BEGSR
C   11                 MOVE 'N'       INSB4              Set for 'after'
C*
C                      MOVEATXT,7     ISTUFF           Save indicatr stuff
C                      ADD  1         CNTR#            Bump counter number
C                      MOVEACNTR#     TXT,93           Append CNTR# to src
C*
C         INSB4        IFEQ 'Y'                        If insert 'before'
C                      EXCPTCSPEC                       Output C-spec
C                      EXCPTPUTSRC                      Put source record
C                      ELSE                            Else insert 'after'
C                      EXCPTPUTSRC                      Put source record
C                      EXCPTCSPEC                       Output C-spec
C                      END                             End IF
C*
C                      ENDSR
C/EJECT
C*
C* Get a source statement from $SOURCE
C*     Set EOF when /* encountered
C*
C         GETSRC       BEGSR
C*
C                      READ $SOURCE                     Read source
C*
C                      MOVEA'    '    TXT,93           Clear pos 93-96
C                      MOVEATXT,1     SLASHA
C         SLASHA       IFEQ '/* '                      If /*
C                      MOVE 'Y'       EOF               Set EOF
C                      END                             End IF
C*
C                      ENDSR
O/EJECT
O*
O* Put a // COPY line
O*
OSRCOUT   E                 PUTCPY
O                                        23 '// COPY LIBRARY-S,NAME-'
```

```
O                                    25 'P$'
O                         PROGNM     31
O*
O* Put a source line
O*
O        E                PUTSRC
O                         TXT        96
O*
O* Fspec for P#xxxxxx file
O*
O        E                FSPEC
O                                     8 '#+FP#'
O                         PROGNM     14
O                                    27 '0           6'
O                                    43 'DISK'
O*
O* Espec for # array
O*
O        E                ESPEC
O                                     6 '#+E'
O                                    20 'P#'
O                         PROGNM     26
O                                    44 '#       1   1  6 0'
O*
O* Cspec for a counter statement
O*
O        E                CSPEC
O                                     6 '#+C'
O                         ISTUFF     17
O                                    44 'ADD  1         #,'
O                         CNTR#      48
```

Figure 16-13

Program PROFL2

```
*    ..  1       . 2         3 .    . 4   . . 5 .  . 6 ..      7         8
H     064                        B       1                             PROFL2
F*
F* RPG Profiler phase 2  set element count in E spec for # array
F* Written by Mel Beckman
F*
FSRCIN   UP  F9600  96            DISK
I*
I* Source input file
I*
ISRCIN
I                                          1   6 SEARCH
I*
I* LDA contains element count
I*
I           UDS
I                                       207 2100CNTR#
C*
C* When E-spec encountered, update it
C*
C           SEARCH    COMP '   #+E'                   LR
OSRCIN   D        LR
O                         CNTR# Z    39
```

Figure 16-14

OCL statements to set Source Required attribute

```
RPGC PROFL1,library,.........,NOHALT,REPLACE,NOLINK,OBJECT    (ten commas)
OLINK PROFL1,library,PROFL1,library,SRQ,,,#RPGLIB
```

Figure 16-15

OCL for executing program SAMP

```
// LOAD SAMP
// FILE NAME-P#SAMP,RECORDS-100,EXTEND-100
// RUN
```

Figure 16-16

Procedure PROFPRT

```
*
* Print a profiled RPG source listing
*
* Parameter 1  Program name (six characters or less)
*           2  Library name
*
// * 'PROFPRT procedure is running'
// LOCAL OFFSET-201,DATA-'?1?',BLANK-6
// LOCAL OFFSET-207,DATA-'?2?',BLANK-8
// LOCAL OFFSET-215,DATA-'?TIME?',BLANK-6   (For 1st page headings)
// LOAD PROFL3
// COMPILE SOURCE-P$?1?,INLIB-?2?
// FILE NAME-$SOURCE,RETAIN-J,BLOCKS-50,EXTEND-100
// FILE NAME-PROFIN,LABEL-P#?1?
// RUN
```

Figure 16-17

Program PROFL3

```
*        1         2         3         4         5         6         7         8
H     064                          8      1                             PROFL3
F*
F* RPG Profiler phase 3  print source listing merged with profile stats
F* Written by Mel Beckman
F*
F* NOTE  This program reads a library source member using the
F*       // COMPILE OCL statement to create a $SOURCE file containing
F*       the member   For this technique to work properly, you must
F*       set the "source required" attribute in the compiled load
F*       member for PROFL1   You do this by compiling and linking the
F*       program in separate steps, using the following OCL
F*
F*  RPGC PROFL3,library,.........,NOHALT,REPLACE,NOLINK,OBJECT (ten commas)
F*  OLINK PROFL3,library,PROFL3,library,SRQ,.,#RPGLIB
F*
F$SOURCE IPE F9600  96            DISK
FPROFIN  ID  F9600   6            DISK
FPRINT   O   F     132     OF     PRINTER
I*
I* Source input file
I*
I$SOURCE
I                                        1  92 TEXT
I                                        1   2 SLASH2
I                                        6   6 TYPE
I                                        4   5 #PLUS
I                                        6   6 TYPE
I                                        7  12 EJECT
I                                       93  96 CNTR#
I*
I* Profile input file
I*
IPROFIN
I                                        1   60COUNT
I*
I* LDA contains the program name, library name, and time of day
I*
I            UDS
I                                      201 206 PROGNM
I                                      207 214 LIBRNM
I                                      215 2200UTIME
C*
C* If tables encountered, quit
C*
```

```
0045 C           SLASH2    COMP '**'                    LR
0046 C   LR                GOTO END
0047 C*
0048 C* Only print original C-spec lines
0049 C*
0050 C                     SETOF                    01      Clear output ind
0051 C           TYPE      IFEQ 'C'                         If a Cspec
0052 C           SLASH2    IFNE '//'                        & not //
0053 C           #PLUS     IFNE '#+'                        & not #+
0054 C           EJECT     IFNE '/EJECT'                     If not EJECT
0055 C                     SETON                    01         Set output ind
0056 C                     Z-ADD0         PCOUNT  60          Clear prt fld
0057 C                     ELSE                              Else no count
0058 C   11                SETON                    OF        Set overflow ind
0059 C                     END                               End IF
0060 C           CNTR#     IFNE *BLANKS                      If counted
0061 C                     READ PROFIN                        Get the counter
0062 C                     Z-ADDCOUNT     PCOUNT  60          Copy to prt fld
0063 C                     END                               End IF
0064 C                     END                              End IF
0065 C                     END                              End IF
0065AC                     SETON                    11
0066 C                     END                              End IF
0067 C*
0068 C* End of job
0069 C*
0070 C           END       TAG
0071 OPRINT   H  103   1P
0072 O       OR  103   OF
0073 O                         UDATE Y    8
0074 O                                   61 'RPG Execution Profile'
0075 O                                   96 'Page'
0076 O                         PAGE  Z  100
0077 OPRINT   H  2     1P
0078 O       OR  2     OF
0079 O                         UTIME      8 '  :  :  '
0080 O                                   32 'Program: xxxxxx'
0081 O                         PROGNM    32
0082 O                                   84 'Library: xxxxxxxx'
0083 O                         LIBRNM    84
0084 OPRINT   D        01
0085 O                         PCOUNTZ    6
0086 O                         TEXT      99
```

Figure 16-18

OCL for compilation

```
RPGC PROFL3,library,,,,,,,,,,NOHALT,REPLACE,NOLINK,OBJECT     (ten commas)
OLINK PROFL3,library,PROFL3,library,SRQ,,,#RPGLIB
```

Figure 16-19

Execution-time cost multipliers for various RPG operations

RPG Operation	Cost milliseconds	Cost multiplier
Indexed disk I/O	100.000	25,000
Nonindexed disk I/O	35.000	8,750
Divide	10.000	2,500
Multiply	5.000	1,250
External program call	3.000	750
Array variable index	1.000	250
Array/table lookup	0.500	125
Other operations	0.004	1

Naming the Compile Listing with the Program Name

by Robert Barber

Code on diskette:
Procedures COMPILE, COMPILEC

I frequently compile several programs at one time, and it is confusing to do a D P and see simply RPGC next to each entry in the spool file. To dispel the confusion of a congested spool, I wrote procedures COMPILE and COMPILEC (Figures 16-20 and 16-21, respectively). These two procedures create a spool entry with the same name, prefixed with a $, as the program being compiled. Thus, I can discern with one glance at the spool file the status of each program being compiled. Procedure COMPILE requires two parameters — program name and library name.

Figure 16-20

Procedure COMPILE

```
// IF ?1?- * ' Enter program to compile'
// IF ?2?- * ' Enter source library name'
// LOCAL OFFSET-1,BLANK-8,DATA-'?1R?'
// LOCAL OFFSET-9,BLANK-8,DATA-'?2R?'
LIBRLIBR ?2?,,P,COMPILEC,$?L'1,7'?,REPLACE
// EVOKE $?L'1,7'? *ALL
```

Figure 16-21

Procedure COMPILEC

```
RPGC ?L'1,8'?,?L'9,8'?,,,XREF,,,?L'9,8'?,,,,+
HALT,,,,?L'9,8'?,NOGEN
REMOVE $?L'1,7?,P,?L'9,8'?
```

Using Indicators Properly in RPG Programs

by Carson A. Soule

Code on diskette:
RPG programs RPGIN1, RPGIN3, RPGIN5

As you've read articles in *NEWS 3X/400*'s fundamentals series or studied the RPG manual, you've surely seen discussions about indicators. Indicators, unique to the RPG programming language, evolved from plugboard concepts used when RPG was first developed more than 25 years ago. A plugboard was programmed using wire jumpers that connected various parts of the computer so signals could be passed.

RPG's indicators replaced the wires with two character symbols — electronic switches you set on or off. You use indicators to store the result of a test, such as a comparison (the indicator is set), and then to condition the execution of the program (the indicator is used). Depending on the type of test for which it is set, an indicator often is thought of as actually indicating the tested condition. In a particular program, indicator 01 may indicate that an employee record was read, and indicator 50 may indicate that earnings were under the FICA limit; however, in a different program, different "meanings" may be assigned to these indicators. Also, some indicators are predefined to test a particular condition; for example, indicator OF is used to test for overflow. As well as replacing the wires of their mechanical predecessor, the plugboard, indicators filled another role—saving program space. When indicators were first used, the entire computer had only 4 K of memory, so programs had to be as short as possible. As late as 1975, RPG programs typically were only 8 K to 12 K. Indicators occupied only one bit of memory each; they made it possible to fit many instructions into a small program.

Indicators also fit the concept of the RPG cycle. Since the program was divided into separate steps—file, input, calculation, and output—a means of communicating between the steps was imperative. Indicators were ideal for this purpose. The creative use of indicators coupled with the RPG cycle resulted in small, efficient programs.

Indicators still use only one bit of memory on the S/36. But with today's larger computers and larger programs, programmers don't need to conserve bits the way they used to. And the efficiency once gained with indicators too often becomes confusion and clutter in present RPG II programs.

With traditional indicator use, each time you encounter an indicator while reading a program, you must determine its meaning. Because an indicator's scope is global and may carry over from cycle to cycle, you may have to hunt through the entire program — or even trace the program's execution — to find out what the indicator means. Avoiding conflicting indicator use when changing code is difficult, and because the meaning of an indicator might depend on code or events far from the point where it is used, reusing indicators becomes dangerous.

Often, whether an indicator is on or off depends on something that happened several cycles ago. Complicated combinations of result and condition indicators create invalid code structures and make even a few lines of code difficult to prove correct. Making small changes to a program's indicators could create major errors in unmodified code, the causes of which would be difficult to pinpoint. Not surprisingly, tracing the flow of execution to find a bug is time-consuming, if not impossible. Misused indicators, then, produce unclear programs without locality (i.e., limiting the indicator use to a small area of the program), and they can result in poorly structured programs.

However, indicators cannot be completely avoided; they are essential to the RPG language. They are the only way to test the results of an I/O operation (chain, read, printer overflow) and to communicate attribute and command key information with screens (SFGR programs). In some cases, indicators are the only efficient way to communicate between calculations and printer output. Indicators also can help you take advantage of the RPG cycle in simple report programs, as indicators were designed to do. The solution is not to abolish indicators, but to use them sparingly and in the least damaging way. *Proper* indicator use is a key to better RPG II programs.

To demonstrate common indicator misuse and alternatives to using indicators, let's compare a program that uses indicators traditionally (Figure 16-22) to a program that uses indicators sparingly (Figure 16-23). You will see as we examine the programs closely that a significant difference between the traditional coding in Figure 16-22 and the more structured coding in Figure 16-23 is that the indicators in Figure 16-23 are not used to communicate between I-specs, C-specs, and O-specs. Limiting indicator use to C-specs makes the second program clearer. You don't need to search for where an indicator is set on or off or puzzle over its meaning.

Figure 16-22's traditionally coded RPG program for a simple file-to-print report (developed expressly for demonstrating some common indicator-use errors) uses the RPG cycle and a wide range of indicators. The I-spec indicators (01, 02, 19, 21, and L1) are the program's first problem. These indicators are scattered throughout the program, far from the code where they were defined; thus, you must remember their meaning to understand the program.

In the C-specs, all the 80 indicators are difficult to understand. Indicator 85 is used on line 16, but is not defined until line 28. Line 32 contains both indicators 81 and 82, creating a compound IF that is difficult to follow.

Whether indicators 83 or 84 (lines 33 through 36) will work correctly if the code is executed more than once is extremely unclear. Line 28 combines indicators 85 and 86 to create a simulated 1P (first page) function that carries over into the O-specs, which are equally difficult to understand because the time line includes five different indicators (lines 43 through 49) to control output. And the last record total (line 66) pulls indicator 45 out of thin air.

Figure 16-22 is just a small example program; imagine this indicator misuse in a program of 2,000 to 3,000 lines! It's easy to see how you can run out of indicators — and patience. Debugging or modifying such a program is an esoteric art of questionable effectiveness.

Figure 16-23 shows an alternative method of coding the program we saw in Figure 16-22. Unnecessary indicators are omitted; the program uses only LR and OF indicators. The program still uses the RPG cycle, although minimally, and takes advantage of the overflow and last record facilities in RPG. The code in Figure 16-23 takes about the same amount of time to write as the code in Figure 16-22 and is dramatically easier to debug, modify, and reuse as the basis for new programs.

The coding in Figure 16-23 removes all the indicators from the I-specs and moves the corresponding functions into the C-specs; moving a test close to the C-specs that depend on it makes a program easier to understand. Not only does this coding simplify indicator use, it eliminates customizing the I-specs, which in turn lets you copy them from a standard definition without modification. The O-specs also are devoid of indicators; the corresponding function is moved to named exception output lines.

In the C-specs, the simulated 1P combination of indicators 85 and 86 (Figure 16-22, lines 16, 28, 29, and 37) is replaced with the field WFST (Figure 16-23, lines 54 and 56). Isolating the time reformatting function code in a separate subroutine (Figure 16-23, FMTTIM, lines 62 through 68) facilitates the function's reuse. Indicators 81 and 82 (Figure 16-22, lines 32, 33, 34, 48, and 49) are replaced with the field WAPM (Figure 16-23, lines 64, 69, and 87). The indicators that were used only in the C-specs and the GOTOs are replaced with IF/ELSE statements.

The advantages of the coding in Figure 16-23 are not restricted to the elimination of indicators; the coding also improves the structure of the program. Eliminating communication indicators in I-specs and O-specs improves the program's locality because individual sections of code are more complete and independent of other parts of the program. All the information needed to understand a section of code is in that section of code, which lets you separate the logic from the function of the program. This separation is most evident in the reusable structured subroutines of the program in Figure 16-23.

Indicators were designed in simpler times, so some of the problems they were designed to solve no longer exist. Today, the poor program structures created by misusing indicators may be hazardous to your programming health. But you can avoid the dangers of indicators by restricting their use to those situations in which they are necessary. Judicious indicator use, in conjunction with structured programming techniques, helps you create more powerful, palatable RPG programs.

Indicator Use: Following the Rules

The following rules and conventions establish a guideline for using indicators correctly.

Never

- use indicators for input field testing (I-specs, columns 65 through 70); instead, use a compare or IF in the C-specs. This alternative puts all the logic in one place, which makes the program easier to read and lets you copy I-specs into the program without modification.
- use indicators for input field relation (I-specs, columns 63 through 64); instead, use a data structure or MOVE in the C-specs. This alternative also puts all the logic in one place.
- use indicators to communicate between cycles (e.g., "first time" or "if any found" indicators); instead, use a field and test it with IF when necessary. A programmer would have to execute the program mentally to understand the meaning of an indicator used for this function.
- use indicators as control switches (e.g., "do once only" or "if security active"); indicators may become corrupted or inconsistent in other parts of the program if used for this purpose. Instead, use a field and test it with IF when necessary, which once again puts the logic in one place.
- use resulting indicators on arithmetic or MOVE instructions. Using a resulting indicator this way hides the implicit IF statement and makes the program difficult to read.
- use SETON or SETOF (except to set a screen indicator) or use a global or catchall SETOF; if you need to use a SETOF, you are using an indicator incorrectly.
- use indicators to communicate between C-spec routines or subroutines; instead, use a control field. Using indicators limits your ability to use the subroutine in another program.
- use indicators to control exception output; instead, use the EXCPT operation with named output lines. The except name is much more comprehensible than an indicator.
- use indicators to condition IF, DO, or END statements — this use sabotages the structure. Comprehending a program that mixes indicators and structured operation codes on the same line is almost impossible. Just ignore the fact that the compiler allows this type of indicator use.

Avoid

- matching records; instead, try to use partial key access (SETLL and READ) or a multi-file input record-out sort. This alternative avoids obscure side-effects involved with MR indicators.
- using indicators on output lines. Control breaks (Lx), Overflow (Ox), and record input

Continued

indicators are acceptable in simple report programs; however, in most cases, exception output conditioned on Lx or Ox is clearer and preferred.

- using indicators in C-specs to implement IF/ELSE, DO, or CAS; instead, use the new structured opcodes if possible. Because RPG II does not support AND and OR for IF/ELSE and DO, indicators are a better solution in some cases. But as soon as the AND/OR option is available, you should never use indicators to implement IF/ELSE, DO, or CAS.

Use a work indicator in calculations

- when sensing end of file, record not found, or an I/O error on READ and CHAIN. This indicator can be stored in a control field if the result is needed later. If the result is used immediately, the work indicator can be used to condition the line immediately following. If more than one line of code needs to be conditioned, use a subroutine or store the result in a field and then test the field with an IF statement.
- on the LOKUP operation to indicate the type of lookup to perform. You can store this indicator in a control field if the result is needed later. If the result is used immediately, you can use the work indicator to condition the line immediately following. If more than one line of code needs to be conditioned, use a subroutine or store the result in a field and then test the field with an IF statement.

If you must use indicators

- to control workstation screens, isolate the indicator to a specific, documented range. Never use the screen indicators to condition C-specs or O-specs.
- when you are using the RPG cycle to generate a report, simplify and document the indicator usage. If you cannot use exception output, limit the O-spec indicators other than control breaks and overflow to a maximum of one per line. Never use the O-spec indicators to condition C-specs. Instead, use IF/ELSE to test the conditions and then SETON the output indicator and document its intent.

Putting Indicators in Their Place

Many experienced programmers have been using RPG's indicators from the day they began programming. Although excluding indicators as much as possible from most of today's programs helps lead to better structured code, eliminating indicators does not in itself make a well-structured program. Poorly structured RPG programs still exist in the indicator-free world of structured programming and in programs created with RPG III. Obviously, indicators are not the only problem, and, if used properly, can benefit RPG programs.

The RPG cycle and built-in functions were designed to make report and file processing programs simple, small, and efficient to write and run. These goals still are desirable if they can be met without giving up good structure; a clear, readable program cannot be sacrificed for dense code. Short report programs, the type of program for which RPG was designed, fit

Continued

naturally into the RPG cycle and let you take advantage of the efficient features of RPG.

The program in Figure 16-24 fits naturally into the RPG cycle. You can relax the rigid principles that ban indicator use in more complex programs and safely use record identifying indicators, L breaks, and overflow to control C-specs and O-specs. To minimize the negative impact of this indicator use, the C-specs use the record identifying indicator only once to execute a processing subroutine.

Using the record indentifying indicators in output lets you use detail output and thereby eliminate exception output and allow overflow use. The C-specs still control output. Note that this is implemented the same way as exception output, with a single communicating indicator set on and off at a single point in calculations. The output lines are limited to one indicator from input (record indentifying indicator or L break) and one from calculations. In all other ways, the program in Figure 16-24 maintains the clarity and structure of Figure 16-23, which uses very few indicators.

These examples demonstrate that indicators themselves have only a small hand in poorly structured RPG programming. The RPG Cycle, L break processing, overflow processing, and other automatic features of RPG also must be used with a great deal of care. To build well-structured, easily read, easily maintained RPG programs, use RPG's automatic features only for the specific functions for which they were designed.

Figure 16-22

Traditional indicator use

```
     *   1 ...   2 ...   3 ...   4 ...   5 ...   6 ...   7 ...   8
000010H                                                          RPGIN
000020F* TRADITIONAL RPGII PROGRAM
000030F* PURPOSE:    PRINT ALL POSITIVE CHECKS AND VOIDS
000040F*             SPACE BETWEEN EMPLOYEES
000050F*             TOTAL ALL PRINTED LINES (SUBTRACTING VOIDS)
000060F*             PRINT  ERROR  IF THE TOTAL IS NEGATIVE
000070FDATAIN  IP  F     128             DISK
000080FPRINTER O   F     132     OF      PRINTER
000090IDATAIN  NS  01      CC
000100I        OR  02      CV
000110I                                  1   1 ARC
000120I                                  2   6OAEMP   L1
000130I                                  7  36 AEMPNM
000140I                                 37  442AERNYR       21
000150I            NS  10
000160C   NTB8BN             EXSR TIME                 TIME AND IP
000170C*
000180C   N01                GOTO VOID
000190C    21                ADD  AERNYR   WERNYR  92 45   TOTAL EARNINGS
000200C*
000210C          VOID        TAG
000220C   N02                GOTO END
000230C    21                SUB  AERNYR   WERNYR          REMOVE VOIDS
000240C*
000250C          END         TAG
000260C* GET TIME AND SIMULATE IP OUTPUT
000270C          TIME        BEGSR                     BEGIN
000280C    B6                SETON                  B5   2ND TIME THRU
000290C    B6                GOTO EXITTM               SKIP CALCS
000300C                      TIME          WTIM    60   HHS  MIN  SEC
000310C                      MOVELWTIM     WHRM    40   HHS  MIN
000320C          WHRM        COMP 1200           B2B1B2AM OR PM
000330C    B1    WHRM        COMP 0 00             B3   MIDNIGHT 1AM
000340C    B2    WHRM        COMP 1300           B4  B4PAST 1 PM
000350C    B3                ADD  1200     WHRM         IF MIDNIGHT
000360C    B4                SUB  1200     WHRM         IF PAST 1 PM
000370C   NB8                SETON                  B8   1ST TIME ONLY
000380C          EXITTM      ENDSR                     END
```

```
000390PRINTER H  104    1P
000400        OR        OF
000410                                      36 'REPORT TITLE'
000420*
000430         D  2     86N85
000440        OR        OF
000450                                       4 'DATE'
000460                          UDATE Y     14
000470                                      20 'TIME'
000480                  81      WHRM        30 '     &AM'
000490                  82      WHRM        30 '     &PM'
000500                                      56 'PAGE'
000510                          PAGE  Z     60
000520*
000530         D 10     L1
000540                          AEMP  Z      5
000550                          AEMPNM      37
000560*
000570         D  1     01 21
000580        OR        02 21
000590                          AERNYRJ     53
000600                                      60 'CHECK'
000610                  02                  60 'VOID '
000620*
000630         T 12     LR
000640                          WERNYRJ     53
000650                                      60 'ERROR'
000660                  45                  60 'TOTAL'
000670*
```

Figure 16-23

Reduced indicator use

```
*         1         2   . . . 3   .     4         5         6         7   . .   8
00001H                                                                    RPGIN3
00002H* INDICATOR FREE RPGII PROGRAM
00003H* PURPOSE   PRINT ALL POSITIVE CHECKS AND VOIDS
00004H*           SPACE BETWEEN EMPLOYEES
00005H*           TOTAL ALL PRINTED LINES (SUBTRACTING VOIDS)
00006H*           PRINT 'ERROR' IF THE TOTAL IS NEGATIVE
00007FDATAIN  IP  F     128            DISK
00008FPRINTER O   F     132      OF    PRINTER
00009IDATAIN  NS
00010I                                      1   1 ARC
00011I                                      2   60AEMP
00012I                                      7  36 AEMPNM
00013I                                     37  442AERNYR
00014C           WINZ      IFEQ ' '                       IF FIRST TIME
00015C                     EXSR FMTTIM                    FORMAT TIME
00016C                     Z-SUB99999     WEMP            INZ EMP L BREAK
00017C                     MOVE 'Y'       WINZ    1       NOT FIRST TIME
00018C                     END                            END IF FIRST TIME
00019C*
00020C           ARC       IFEQ 'C'                       IF CHECK RCD
00021C                     EXSR PRTHDG                    PRINT HEADINGS
00022C                     EXSR PRTEMP                    PRINT EMPLOYEE
00023C           AERNYR    IFGT 0                         IF EARNING>0
00024C                     ADD  AERNYR    WERNYR  92      TOTAL EARNINGS
00025C                     EXCPTPRTCHK                    PRINT CHECK
00026C                     END                            END IF EARNINGS
00027C                     END                            END IF CHECK
00028C*
00029C           ARC       IFEQ 'V'                       IF VOID RCD
00030C                     EXSR PRTHDG                    PRINT HEADINGS
00031C                     EXSR PRTEMP                    PRINT EMPLOYEE
00032C           AERNYR    IFGT 0                         IF EARNING>0
00033C                     SUB  AERNYR    WERNYR          REMOVE VOIDS
00034C                     EXCPTPRTVOD                    PRINT VOID
00035C                     END                            END IF EARNINGS
00036C                     END                            END IF VOID
00037C*
00038CLR         WERNYR    IFGE 0                         IF TOTAL->0
00039CLR                   EXCPTPRTLR                     PRINT LR
00040CLR                   ELSE                           END IF TOTAL
00041CLR                   EXCPTPRTERR                    PRINT LR ERROR
00042CLR                   END                            END IF TOTAL
```

```
00043C*
00044C* PRINT EMPLOYEE ON NEW EMPLOYEE NUMBER
00045C           PRTEMP    BEGSR                                PRINT EMPLOYEE
00046C           AEMP      IFNE WEMP                            IF NEW EMPLOYEE
00047C                     EXCPTRPTEMP                          PRINT EMPLOYEE LINE
00048C                     Z-ADDAEMP      WEMP    50            SAVE NEW EMP#
00049C                     END                                  END IF NEW EMP
00050C                     ENDSR                                END PRT EMP
00051C*
00052C* PRINT HEADINGS ON FIRST PAGE AND OVERFLOW
00053C           PRTHDG    BEGSR                                PRINT HEADINGS
00054C           WFST      IFEQ ' '                             IF FIRST TIME
00055C                     EXCPTRPTHDG                          PRINT HEADING
00056C                     MOVE 'N'       WFST    1             NOT FIRST TIME
00057C                     ELSE                                 ELSE NOT FIRST
00058C   OF                EXCPTRPTHDG                          PRINT HEADING
00059C                     END                                  END IF FIRST
00060C                     ENDSR                                END PRINT HDG
00061C*
00062C* RETRIEVE AND FORMAT THE TIME
00063C           FMTTIM    BEGSR                                FORMAT TIME
00064C                     MOVE 'AM'      WAPM    2             ASSUME AM
00065C                     TIME           WTIM    60            RETRIEVE TIME
00066C                     MOVELWTIM      WHRM    40            HOURS & MINUTES
00067C           WHRM      IFGE 1200                            IF >= NOON
00068C           WHRM      IFLT 2400                            AND < MIDNIGHT
00069C                     MOVE 'PM'      WAPM    2             THEN PM
00070C                     END                                  END IF NOON
00071C                     END                                  END IF MIDNT
00072C           WHRM      IFGE 1300                            IF >= 1:00PM
00073C                     SUB  1200      WHRM                  ADJUST HOURS
00074C                     END                                  END IF PM
00075C           WHRM      IFLT 0100                            IF < 1:00AM
00076C                     ADD  1200      WHRM                  ADJUST HOURS
00077C                     END                                  END IF PM
00078C                     ENDSR                                END FMT TIM
000790PRINTER E  104           RPTHDG
000800                                   50 'REPORT TITLE'
000810*
000820        E  2             RPTHDG
000830                                    4 'DATE'
000840                         UDATE Y   14
000850                                   20 'TIME'
000860                         WHRM      27 ' 0  '
000870                         WAPM      30
000880                                   56 'PAGE'
000890                         PAGE  Z   60
000900*
000910        E 10             RPTEMP
000920                         AEMP  Z    5
000930                         AEMPNM    37
000940*
000950        E  1             RPTCHK
000960                         AERNYRJ   53
000970                                   60 'CHECK'
000980*
000990        E  1             RPTVOD
001000                         AERNYRJ   53
001010                                   59 'VOID'
001020*
001030        E 12             RPTLR
001040                         WERNYRJ   53
001050                                   60 'TOTAL'
001060*
001070        E 12             RPTERR
001080                         WERNYRJ   53
001090                                   60 'ERROR'
001100*
```

Figure 16-24

Better indicator use

```
*         1         2 .       3         4         5         6         7         8
00001H                                                                      RPGIN5
00002H* S/36 INTELLIGENT INDICATOR RPGII PROGRAM
00003H* PURPOSE   PRINT ALL POSITIVE CHECKS AND VOIDS.
00004H*           SPACE BETWEEN EMPLOYEES
00005H*           TOTAL ALL PRINTED LINES (SUBTRACTING VOIDS)
00006H*           PRINT 'ERROR' IF THE TOTAL IS NEGATIVE
00007FDATAIN  IP  F     128            DISK
00008FPRINTER O   F     132     OF     PRINTER
00009IDATAIN  NS  01   1 CC
00010I        OR  02   1 CV
00011I                                        1   1 ARC
00012I                                        2   60AEMP  L1
00013I                                        7  36 AEMPNM
00014I                                       37  442AERNYR
00015I        NS  19
00016C                    EXSR PRTHDG                     PRINT HEADINGS
00017C   01               EXSR PRCCHK                     PROCESS CHECK RECORD
00018C   02               EXSR PRCVOD                     PROCESS VOID RECORD
00019C*
00020CLR          WERNYR    COMP 0                    34  34IF TOTAL->O LR ELSE ERROR
00021C*
00022C* PROCESS CHECK RECORD
00023C          PRCCHK    BEGSR                           PROCESS CHECK
00024C          AERNYR    IFGT 0                          IF EARNING>0
00025C                    ADD  AERNYR    WERNYR  92       TOTAL EARNINGS
00026C                    SETON                     32    PRINT CHECK
00027C                    ELSE                            ELSE
00028C                    SETOF                     32    NO PRINT
00029C                    END                             END IF
00030C                    ENDSR                           END PROCESS CHK
00031C*
00032C* PROCESS VOID RECORD
00033C          PRCVOD    BEGSR                           PROCESS VOID
00034C          AERNYR    IFGT 0                          IF EARNING>0
00035C                    SUB  AERNYR    WERNYR           REMOVE VOIDS
00036C                    SETON                     33    PRINT VOID
00037C                    ELSE                            ELSE
00038C                    SETOF                     33    ELSE NO PRINT
00039C                    END                             END IF
00040C                    ENDSR                           END PROCESS VOID
00041C*
00042C* PRINT HEADINGS ON FIRST PAGE AND OVERFLOW
00043C          PRTHDG    BEGSR                           PRINT HEADINGS
00044C          WFST      IFEQ ' '                        IF FIRST TIME
00045C                    EXSR FMTTIM                     FORMAT TIME
00046C                    MOVE 'N'       WFST    1        NOT FIRST TIME
00047C                    SETON                     30    PRINT HEADING
00048C                    ELSE                            ELSE
00049C                    SETOF                     30    ELSE NOPRINT
00050C                    END                             END IF
00051C                    ENDSR                           END PRINT HDG
00052C*
00053C* RETRIEVE AND FORMAT THE TIME
00054C          FMTTIM    BEGSR                           FORMAT TIME
00055C                    MOVE 'AM'      WAPM    2        ASSUME AM
00056C                    TIME           WTIM    60       RETRIEVE TIME
00057C                    MOVELWTIM      WHRM    40       HOURS & MINUTES
00058C          WHRM      IFGE 1200                       IF >= NOON
00059C          WHRM      IFLT 2400                       AND < MIDNIGHT
00060C                    MOVE 'PM'      WAPM    2        THEN PM
00061C                    END                             END IF NOON
00062C                    END                             END IF MIDNT
00063C          WHRM      IFGE 1300                       IF >= 1:00PM
00064C                    SUB  1200      WHRM             ADJUST HOURS
00065C                    END                             END IF PM
00066C          WHRM      IFLT 0100                       IF < 1:00AM
00067C                    ADD  1200      WHRM             ADJUST HOURS
00068C                    END                             END IF PM
00069C                    ENDSR                           END FMT TIM
00070OPRINTER D  104    30
00071O        OR        OF
00072O                                     50 'REPORT TITLE'
00073O*
00074O        D  2      30
00075O        OR        OF
```

```
                                   4 'DATE'
                         UDATE Y  14
                                  20 'TIME'
                         WHRM     27 ' 0.  '
                         WAPM     30
                                  56 'PAGE'
                         PAGE  Z  60
*
          O 10      L1
                         AEMP  Z   5
                         AEMPNM   37
*
          O  1      01 32
                         AERNYRJ  53
                                  60 'CHECK'
*
          D  1      02 33
                         AERNYRJ  53
                                  59 'VOIO'
*
          T 12      LR 34
                         WERNYRJ  53
                                  60 'TOTAL'
*
          T 12      LRN34
                         WERNYRJ  53
                                  60 'ERROR'
*
```

Saving and Restoring Indicators, Part 1

by John Field

Importing large blocks of code or subroutines into existing RPG source can result in the multiple use of indicators, which can cause debugging headaches when you try to make sure that setting an indicator on or off in one part of the program does not affect another part of the same program adversely. Subroutines SAVIND and RSTIND (Figure 16-25) can eliminate this housekeeping problem.

The concept of using subroutines SAVIND and RSTIND is simple. Before a program executes a subroutine or block of code that might use existing indicators, the current values of these indicators are saved, and then the indicators are initialized (SETOF). After the subroutine or block of code has been executed, the indicators are reset to their previous values.

In the example, assume that a subroutine (SUB001) is imported into an existing program. The imported subroutine uses indicators 01, 02, 03, 15, 16, 50, 51, 70, and 71. (To determine which indicators a subroutine uses, copy the subroutine into a separate RPG source member and then run the S/36 RPGC procedure and check the indicator summary.) To save the existing program's indicators, subroutine SAVIND is executed immediately before subroutine SUB001 is executed. To restore the indicators to their original values, subroutine RSTIND is executed immediately after subroutine SUB001. As far as the rest of the program is concerned, the indicator values remain unchanged.

The same technique can be used if you import a block of code into an existing program. Simply execute subroutine SAVIND before the block of code, and execute subroutine RSTIND after the block of code. This method has saved me hours of debugging time when I have been working with large, complicated programs.

Figure 16-25

Subroutines SAVIND and RSTIND

```
*     . 1         2 .   . . 3 . .  .   4      .  5         6      .  7         8
C*--------------------
C                    EXSR SAVIND
C                    EXSR SUB001
C                    EXSR RSTIND
C*--------------------


C* SUBROUTINE  SAVIND
C*         SAVES CURRENT VALUES OF INDICATORS AND
C*                SETS THESE INDICATORS OFF
C*
C          SAVIND    BEGSR
C                    MOVE 0         IN01    10
C                    MOVE 0         IN02    10
C                    MOVE 0         IN03    10
C                    MOVE 0         IN15    10
C                    MOVE 0         IN16    10
C                    MOVE 0         IN50    10
C                    MOVE 0         IN51    10
C                    MOVE 0         IN70    10
C                    MOVE 0         IN71    10
C   01               MOVE 1         IN01
C   02               MOVE 1         IN02
C   03               MOVE 1         IN03
C   15               MOVE 1         IN15
C   16               MOVE 1         IN16
C   50               MOVE 1         IN50
C   51               MOVE 1         IN51
C   70               MOVE 1         IN70
C   71               MOVE 1         IN71
C                    SETOF                     010203
C                    SETOF                     151650
C                    SETOF                     517071
C                    ENDSR
C*
C*
C* SUBROUTINE  RSTIND
C*                   INDICATORS ARE RESET TO THEIR ORIGINAL
C*                   VALUES
C*
C*
C          RSTIND    BEGSR
C          IN01      COMP 1                    01
C          IN02      COMP 1                    02
C          IN03      COMP 1                    03
C          IN15      COMP 1                    15
C          IN16      COMP 1                    16
C          IN50      COMP 1                    50
C          IN51      COMP 1                    51
C          IN70      COMP 1                    70
C          IN71      COMP 1                    71
C                    ENDSR
C*
```

Saving and Restoring Indicators, Part 2

by Ron Elliott

Saving and Restoring Indicators, Part 1 (page 541) demonstrates a technique for saving and subsequently restoring the status of RPG indicators. In Figure 16-26, I present an alternative method that accomplishes the same thing with a marked reduction in the length of the source program, the amount of main storage required, and the number of library sectors required for the object program.

John's program uses the MOVE and COMP operations and a one-byte field to retain the status of each indicator. The technique illustrated in Figure 16-26 uses the BITON, BITOF, and TESTB operations, which allow the use of only one bit per indicator.

In Figure 16-26, one BITOF operation initializes the status of all eight bits in the one-byte variable INDS. (If you need to initialize more than eight bits, you need to use multiple BITOF operations.) After the bits are initialized, multiple BITON operations set on bits in variable INDS, according to the status of the indicators. Thus, bit 0 is set on if indicator 01 is on, bit 1 is set on if indicator 02 is on, and so on through the necessary indicators.

Then, as in John's technique, all the indicators are set off preparatory to the execution of an imported subroutine or block of code. After the imported code is executed, multiple TESTB operations restore the status of the original indicators depending on the bit settings in field INDS.

Figure 16-26

Technique for saving and restoring indicators

```
* .. . . 1 .. . . 2 ... .. 3 .... . 4 . . ... 5 ... ... 6 ... ... 7 ... .. 8
C                    BITOF'01234567'INDS     1
C    01              BITON'0'       INDS
C    02              BITON'1'       INDS
C    03              BITON'2'       INDS
C    15              BITON'3'       INDS
C    16              BITON'4'       INDS
C    50              BITON'5'       INDS
C    51              BITON'6'       INDS
C    70              BITON'7'       INDS
C                    SETOF                       010203
C                    SETOF                       151650
C                    SETOF                       5170
C*
C* Execute imported subroutine or block of code
C*
C                    TESTB'0'       INDS             01
C                    TESTB'1'       INDS             02
C                    TESTB'2'       INDS             03
C                    TESTB'3'       INDS             15
C                    TESTB'4'       INDS             16
C                    TESTB'5'       INDS             50
C                    TESTB'6'       INDS             51
C                    TESTB'7'       INDS             70
```

Reversing the Value of an Indicator of an Unknown Status

by Ron Elliott

RPG programmers often want to reverse the unknown status of an indicator. That is, if indicator 10 is on, they want to set it off; but if it is off, they want to set it on. The first impulse is to code:

```
N10 SETON 10
 10 SETOF 10
```

But, like so many impulses, this is not a good idea. The first line will set the indicator on if it is off, but then the second line will just set it back off again. Reversing the sequence of these two lines doesn't help either — the other way results in 10 always being on.

There are a number of ways to solve this problem, but one of the easiest involves using an additional indicator. The sequence

```
   SETON 11
10 SETOF 1011
11 SETON 10
```

will produce the desired result with a minimum of fuss.

Checking an Indicator in an IF Statement

by Wells Cooner

How many times have you wanted the ability on the S/36 to check the status of an indicator with an IF statement so you could execute a section of code without having to put the indicator on each line of code? On the S/38, you can use an indicator in an IF statement by specifying the field *IN*xx* (where *xx* is the indicator to be tested) as one of the compare fields. To simulate the same function on the S/36, consider the code in Figure 16-27.

To understand how this example works, remember two things. First, an IF statement can be conditioned by an indicator. If the indicator is on, the IF condition is checked. If the indicator is not on, the IF statement and all the code that would be executed by the IF statement is bypassed. Second, it is always true that a blank is equal to a blank. Therefore, in Figure 16-27, if indicator 98 is on, the code after the IF statement is executed. If indicator 98 is off, the code is bypassed. If you want to check multiple indicators, you can add the appropriate OR statements or add IF statements within the same group.

Figure 16-27

Checking an indicator in an IF statement.

```
*      1         2         3         4         5         6         7         8
     C                     CHAINDATAFILE             98
     C   98        ' '     IFEQ ' '
     C                     MOVE 'E'       ERROR
     C                     GOTO TAGEND
     C                     END
     C                     MOVE 'E'       ERROR
     C                     GOTO TAGEND
```

Nesting IF Statements

answered by Ron Elliott

Q I've successfully written RPG II structured programs. The results I get from the code in Figure 16-28, however, puzzle me, and the RPG manual offers no help. The code is supposed to direct a program to perform the THEN clause (lines 14 and 15) when all IF conditions are met. When any of the stated conditions fail, the program should perform the ELSE clause (lines 17 and 18). But sometimes my program doesn't perform either the THEN or the ELSE clause. Why not?

A Your programs perform neither clause because your code is incorrect. In any language, an ELSE statement refers to the most recent IF preceding it. To relate the multiple IF conditions tested by your series of nested IF statements to the THEN and ELSE statements, you need an AND statement, which is an RPG III language feature that does not exist in RPG II.

In the code in Figure 16-28, the THEN clause is executed when all the stated conditions are met. However, the ELSE clause is executed only when the *last* IF statement (line 13) fails. Should any of the IFs before the last one fail, your program bypasses both the THEN and the ELSE clauses.

You can solve your problem in one of two ways. One way is to change the nested IF statements to COMP statements with chained indicators to create an "and" relationship between conditions (Figure 16-29). The other way is to use the code shown in Figure 16-30, which causes the THEN clause to be executed as it is in Figure 16-29 (i.e., when all conditions are met). The ELSE clause's execution criteria in Figure 16-30, however, is slightly different from that in Figure 16-29. Instead of the ELSE clause being executed only when the last condition fails, the ELSE clause is now executed when the THEN clause does not happen.

Figure 16-28

Incorrect nesting of IF statements for cumulative effect

```
*        1         2         3         4         5         6         7        . 8
0001 C**************************************************************************
0002 C*       SUB-ROUTINE "ACCUM," WHICH ADDS RECORD'S SALES INTO "TOTSLS"
0003 C*       ONLY WHEN SEVERAL CONDITIONS ARE MET   IF NOT MET, ADD
0004 C*       RECORD'S SALES INTO "BADSLS "
0005 C**************************************************************************
0006 CSR         ACCUM     BEGSR
0007 CSR                   MOVE 'XXXX'    CHECK   4
0008 CSR         HAREA     IFGE AREAFM
0009 CSR         HAREA     IFLE AREATO
0010 CSR         HTERR     IFGE TERRFM
0011 CSR         HTERR     IFLE TERRTO
0012 CSR         HTYPE     IFGE TYPEFM
0013 CSR         HTYPE     IFLE TYPETO
0014 CSR         TOTSLS    ADD  RECSLS    TOTSLS  72
0015 CSR                   MOVE 'GOOD'    CHECK
0016 CSR                   ELSE
0017 CSR                   ADD  PECSLS    BADSLS  72
0018 CSR                   MOVE 'BADX'    CHECK
0019 CSR                   END
0020 CSR                   END
0021 CSR                   END
0022 CSR                   END
0023 CSR                   END
0024 CSR                   END
0025 C*                                    BADSLS
0026 CSR          'CHECK'  DEBUGPRINTER    CHECK
0027 CSR                   ENDSR
```

Figure 16-29

Replacing nested IF statements with COMP statements and chained conditions

```
*... .   1  .   .. 2 ... ... 3 ..     4  .. ... 5  .. .. 6     .. 7 ..  ... 8
                 HAREA     COMP AREAFM                 11  11
           11    HAREA     COMP AREATO                   1111
           11    HTERR     COMP TERRFM                 11  11
           11    HTERR     COMP TERRTP                   1111
           11    HTYPE     COMP TYPEFM                 11  11
           11    HTYPE     COMP TYPETO                   1111
           11              ADD  RECSLS    TOTSLS
          N11              ADD  RECSLS    BADSLS
```

Figure 16-30

Linking the execution of the ELSE clause with the THEN clause

```
*... ... 1 ..  . . 2     . . 3 .   . . 4 ... ... 5 ..     6  .  . . 7  ... .  8
0001 C                     MOVE 'XXXX'    CHECK   4
0002 C           HAREA     IFGE AREAFM
0003 C           HAREA     IFLE AREATO
0004 C           HTERR     IFGE TERRFM
0005 C           HTERR     IFLE TERRTP
0006 C           HTYPE     IFGE TYPEFM
0007 C           HTYPE     IFLE TYPETO
0008 C                     ADD  RECSLS    TOTSLS  72
0009 C                     MOVE 'GOOD'    CHECK
0010 C                     END
0011 C                     END
0012 C                     END
0013 C                     END
0014 C                     END
0015 C                     END
0016 C            CHECK    IFNE 'GOOD'
0017 C                     ADD  RECSLS    BADSLS  72
0018 C                     MOVE 'BADX'    CHECK
0019 C                     END
```

Printing Action Diagrams for Structured Verbs

by Gary T. Kratzer

program by Steve Cranmer

Code on diskette:
Procedure NEST
RPG program NEST
Screen format member NESTFM

Using the RPG structured operations IF/ELSE, DO, DOU*xx* (do until), DOW*xx* (do while), and CAS can be a mixed blessing. On one hand, the structured verbs can dramatically reduce indicator use and improve program readability. On the other hand, structured verbs can easily create a mass of spaghetti code faster than you can say "top-down programming." At best, spaghetti code can make reading a program a laborious task; matching a structured operation with its associated END statement is difficult because you can "nest" your logic to essentially an infinite number of levels. This process can be especially painful if the program you are attempting to debug is not your own or is one you haven't looked at for a while.

To make life in the structured programming world a little easier, we offer utility NEST, which can produce action diagrams up to 16 nesting levels deep for structured verbs. The resulting action diagrams can be inserted directly into your RPG program, printed as a report, or both. To create the NEST utility, first create procedure NEST, compile program NEST, and compile screen format member NESTFM.

A prompt screen appears (Figure 16-31; see Figure 16-32 for screen format member NESTFM) to request the required input parameters when you call procedure NEST (Figure 16-33). The first parameter is the program to be diagrammed, the second is the library in which the program resides, the third indicates whether you want a printed copy of the diagrammed source, and the fourth indicates whether you want the original source program updated with the action diagrams. To cancel the procedure from the prompt screen, press Command key 3.

Figure 16-31
NEST prompt screen

```
Diagram Conditional RPG Statements

Enter the source member name                        ______

Enter the library name of the member            .   ______

Print diagrammed copy of program.(Y,N)              Y

Update source program directly (Y,N)                N



Cmd3-Exit
```

After you supply all the parameters, press Enter, and the program to be diagrammed is copied via the $MAINT utility to a 96-byte work file named ?WS?NESTWK. Then, program NEST is loaded to perform the actual diagramming.

Program NEST (Figure 16-34) reads the work file and looks for the IF, ELSE, CAS, DO, DOU*xx*, DOW*xx*, and END operations. When it finds one of these operations, program NEST increments or decrements a counter that indicates the current nest level and places action diagram symbols on that line in the appropriate position based on the level. The action diagrams occupy positions 80 through 96 of the source statement line. Each time program NEST inserts a diagram symbol into a line, it updates the work file if you requested that the source be updated. Be aware that program NEST does not check for existing comments in positions 80 through 96 before inserting the diagram symbols. If program NEST encounters an ELSE statement, it places a left arrow symbol (<) on that line as an additional aid to spotting those statements.

When program NEST finishes processing the target program, if you requested that your source be updated with the action diagrams, procedure NEST calls $MAINT again to replace the existing program in the library with the updated version from the ?WS?NEST work file. Also, if you requested a printout of the source, program NEST sends it to the spool queue. For a sample of the action diagrams program NEST produces, take a closer look at program NEST itself in Figure 16-34. Notice that columns 80 through 96 contain the action diagrams for the structured operations used in the program.

Program NEST diagrams a maximum of 16 levels, which should suffice for almost any well-structured program, but you could make some minor modifications to increase the nest level. Another useful feature you could add would be to diagram GOTO and TAG statements because these are related statements. So the next time spaghetti code tries to ruin your debugging efforts, pull utility NEST out of your programming arsenal and get the job done right.

Figure 16-32

Screen format member NESTFM

```
*... ... 1 ... ... 2 ... ... 3 ... .. 4 ... .. 5 . . . .. 6 ... ... 7 ... ... 8
SNEST      124      YY         Y                       C
D         34 422Y                Y        Y         CDiagram Conditional RPGX
D Statements
D         49 713Y                                   CEnter the source memberX
D name ........... ... ....
DSCRFMT    8 763Y  Y     Y       Y        Y
D         49 913Y                                   CEnter the library name X
Dof the member . ........ .
DLIBNAM    8 963Y  Y             Y        Y
D         491113Y                                   CPrint diagrammed copy oX
Df program (Y,N) ..... ....
DPRINT     11163Y  Y             Y        Y
D         491313Y                                   CUpdate source program dX
Directly (Y,N)  .......  ..
DUPDATE    11363Y  Y             Y        Y
D         1423 4Y                                   CCmd3-Exit
```

Figure 16-33

Procedure NEST

```
// EVALUATE P3-'Y' P4-'N'
// IF ?2?-  EVALUATE P2-?CLIB?
// IF ?1?-  PROMPT MEMBER-NESTFM,FORMAT-NEST,LENGTH-'8,8,1,1'
// IF ?CD?-2003 RETURN
// IF ?3?-N IF ?4?-N RETURN
**
// * '?1?,?2? now being diagrammed'
// LOCAL OFFSET-1,BLANK-18,DATA-'?1?'
// LOCAL OFFSET-9,DATA-'?2?'
// LOCAL OFFSET-17,DATA-'?3??4?'
// LOAD $MAINT
// FILE NAME-?WS?NESTWK,RECORDS-1000,EXTEND-100
// RUN·
// COPY OMIT-SYSTEM,NAME-?1?,LIBRARY-S,FROM-?2?,TO-DISK,FILE-?WS?NESTWK,RECL-96
// END
**
// LOAD NEST
// FILE NAME-NESTWK,LABEL-?WS?NESTWK
// RUN
**
```

```
// IF ?4?=N GOTO END
// LOAD $MAINT
// FILE NAME-?WS?NESTWK,RETAIN-S
// RUN
// COPY NAME-?1?,LIBRARY-S,FROM-DISK,TO-?2?,FILE-?WS?NESTWK,RETAIN-R
// END
// TAG END
// IF DATAF1-?WS?NESTWK DELETE ?WS?NESTWK,F1
**--------------------------------
** PROCEDURE     - NEST
** AUTHORS       - STEPHEN C  CRANMER   INDUSTRIAL TRAINING SYSTEMS, MARLTON, NJ
** COMMENTS      - DIAGRAM CONDITIONAL RPG STATEMENTS
**--------------------------------
```

Figure 16-34

Program NEST

```
*        1         2         3         4         5         6         7         8
H      64                          8       1                              NEST
F**--------------------------------
F** PROGRAM ID - NEST
F** AUTHOR      - STEPHEN C  CRANMER   INDUSTRIAL TRAINING SYSTEMS
F** COMMENTS    - SET DIAGRAMS ON STRUCTURED CONDITIONAL CODE
F**             - DIAGRAMS UP TO 16 NESTED LEVELS
F**             - ASSUMES STRUCTURED OPERATORS ARE MATCHED PROPERLY
F**               AND PROGRAM WILL COMPILE SUCCESSFULLY
F**             - OPTIONS FROM THE LDA TO PRINT AND/OR UPDATE SOURCE
F** INDICATORS - OF - PAGE OVERFLOW
F**--------------------------------
F** NEST PROGRAM WORK FILE
FNESTWK  UP  F9600  96            DISK
F** PRINTER FILE IF PRINT-YES
FPRINTER O   F 132 132     20F      PRINTER
E                   RH       1   3 85              HEADINGS FOR PRINTOUT
E                   LVL      1  16 16              NESTED LEVEL SYMBOLS
E                   CL          16  1              CURRENT LINE
INESTWK
I                                        1  96 STMT                RPG STATEMENT
I                                        1   7 $MAINT              $MAINT CONTROLS
I                                        1   3 BEGTAB              TABLE/ARRAY STARTED
I                                        6   6 SPEC                SPEC TYPE
I                                        7   7 ASTRSK              TEST FOR COMMENT
I                                       28  29 IF#DO               TEST FOR IF, DO
I                                       28  30 CAS                 TEST FOR CASXX
I                                       28  32 OPER                TEST FOR ELSE, END
I** DATA STRUCTURE FOR SYSTEM TIME/DATE
I            DS
I                                        1  120SYSDS               SYSTEM TIME/DATE
I                                        1   40SYSHM               SYSTEM HOUR/MIN
I                                        7  120SYSMDY              SYSTEM MM/DD/YY
I** LOCAL DATA AREA TO PICK UP OPTIONS
I           UDS
I                                        1   8 MEMBER              MEMBER NAME
I                                        9  16 FRMLBR              FROM LIBRARY
I                                       17  17 PRINT               PRINT Y,N
I                                       18  18 UPDATE              UPDATE Y,N
C**--------------------------------
C** FIRST CYCLE PROCESSING
C           FIRST     IFEQ *BLANK                      FIRST CYCLE?          ---------------+
C                     MOVE 'N'       FIRST   1         FIRST CYCLE FLAG                     |
C                     Z-ADD16        C      20         COLUMN MARKER                        |
C                     MOVEA*BLANKS   CL                BLANK CURRENT LINE                   |
C                     MOVE *BLANKS   OUT    16         BLANK OUTPUT AREA                    |
C**                                                                                         |
C           PRINT     IFEQ 'Y'                         PRINT-YES?            --------------+|
C                     TIME           SYSDS             SYSTEM TIME/DATE                    ||
C                     MOVE 'H'       PRIOR   1         TEST FOR SPEC CHANGE                ||
C                     MOVE 'N'       TAFLAG  1         TABLE/ARRAY FLAG                    ||
C                     EXCPTHDNGS                       PRINT HEADINGS PG-1                 ||
C                     END                                                    --------------+|
C**                                                                                         |
C                     END                                                    ---------------+
```

```
C**
C** PROCESS CALC SPECS ONLY
C          SPEC      IFEQ 'C'                          CALC SPEC?          ---------------+
C**                                                                                       |
C** CLEAN OUT '<' CODE FROM PREVIOUS ELSE LINE                                             |
C          ELSFLG    IFEQ 'Y'                          ELSE-SEGMENT ACTIVE? --------------+|
C                    MOVE ' '       ELSFLG  1          RESET FLAG                         ||
C          C         IFGT *ZERO                        LESS THAN 16 LEVELS  -------------+||
C                    MOVE ' '       CL,C               BLANK OUT '<'                     |||
C                    END                                                    -------------+||
C                    END                                                    --------------+|
C**                                                                                       |
C          ASTRSK    IFNE '*'                          NON-COMMENT?        --------------+|
C**                                                                                      ||
C** MOVE CONTINUATION LINE TO OUTPUT AS DEFAULT                                          ||
C                    MOVEACL        OUT                SET DEFAULT OUTPUT                ||
C**                                                                                      ||
C** PROCESS END OF CONDITIONAL SEGMENT, MOVING RIGHT ONE COLUMN                          ||
C          OPER      IFEQ 'END  '                      END STATEMENT?      -------------+||
C                    MOVE ' '       CASFLG             BLANK CASE FLAG                  |||
C                    ADD  1         C                  MOVE RIGHT                       |||
C          C         IFGT *ZERO                        LESS THAN 16 LEVELS ------------+|||
C                    MOVEALVL,C     OUT                FILL OUTPUT AREA                ||||
C                    MOVE ' '       CL,C               BLANK POSITION                  ||||
C                    END                                                   ------------+|||
C                    END                                                   -------------+||
C**                                                                                      ||
C** PROCESS ELSE STATEMENT WITH '<' MARKER                                               ||
C          OPER      IFEQ 'ELSE '                      ELSE STATEMENT?     -------------+||
C          C         IFGT *ZERO                        LESS THAN 16 LEVELS ------------+|||
C                    MOVE 'Y'       ELSFLG  1          SET ELSE-FLAG                   ||||
C                    MOVE '<'       CL,C               SET SYMBOL                      ||||
C                    MOVEACL        OUT                FILL OUTPUT AREA                ||||
C                    END                                                   ------------+|||
C                    END                                                   -------------+||
C**                                                                                      ||
C** PROCESS FIRST CASE STATEMENT IN A SERIES, MOVING LEFT ONE COLUMN                     ||
C          CAS       IFEQ 'CAS'                        CASXX STATEMENT     -------------+||
C          CASFLG    IFEQ ' '                          NOT ALREADY ACTIVE  ------------+|||
C                    MOVE 'Y'       CASFLG  1          SET CASE FLAG                   ||||
C                    EXSR NXTLVL                                                       ||||
C                    END                                                   ------------+|||
C                    END                                                   -------------+||
C**                                                                                      ||
C** PROCESS IF/DO STATEMENTS, MOVING LEFT ONE COLUMN                                     ||
C          IF#DO     CASEQ'IF'      NXTLVL             IFXX STATEMENT      -------------+||
C          IF#DO     CASEQ'DO'      NXTLVL             DOXXX STATEMENT                  |||
C                    END                                                   -------------+||
C**                                                                                      ||
C** PROCESS CALC COMMENT STATEMENTS WITH CURRENT LINE                                    ||
C                    ELSE                                                               <||
C                    MOVEACL        OUT                FILL OUTPUT AREA                  ||
C                    END                                                   --------------+|
C**                                                                                       |
C** UPDATE ALL CALC SPECS WITH CURRENT LINE IF UPDATE-YES                                 |
C          UPDATE    IFEQ 'Y'                          UPDATE SOURCE?      --------------+|
C                    EXCPTUPDREC                       UPDATE SOURCE MEMBER              ||
C                    END                                                   --------------+|
C**                                                                                       |
C                    END                                                   ---------------+
C**
C** PRINT SOURCE PROGRAM IF PRINT-YES
C          PRINT     IFEQ 'Y'                          PRINT SOURCE?       ---------------+
C          $MAINT    IFNE '// COPY'                    NOT CONTROL SPEC    --------------+|
C          $MAINT    IFNE '// CEND'                    NOT CONTROL SPEC    -------------+||
C   OF               EXCPTHDNGS                        PRINT HEADINGS                   |||
C**                                                                                     |||
C** CHECK FOR CHANGE IN TYPE OF SPEC FOR DOUBLE SPACING                                  |||
C          SPEC      IFNE PRIOR                        NEW SPEC TYPE?      ------------+|||
C          TAFLAG    IFEQ 'N'                          TABLES NOT STARTED  -----------+||||
C                    EXCPTSKIP                         PRINT BLANK LINE               |||||
C                    END                                                   -----------+||||
C                    END                                                   ------------+|||
C                    MOVE SPEC      PRIOR              SET FOR NEXT RECORD              |||
C**                                                                                     |||
C** SET FLAG FOR COMPILE TIME TABLE/ARRAY START (STOP DOUBLE SPACING)                    |||
```

```
0131 C           BEGTAB    IFEQ '** '                      TABLE/ARRAY START    ------------+|||
0132 C                     MOVE 'Y'       TAFLAG           TABLE/ARRAY FLAG                 ||||
0133 C                     END                                                  ------------+|||
0134 C**                                                                                     |||
0135 C** PRINT NEST SYMBOLS ONLY ON CALC SPECS                                               |||
0136 C           SPEC      IFEQ 'C'                        CALC SPEC?           ------------+|||
0137 C                     EXCPTPRNEST                     PRINT STMT W/NEST                ||||
0138 C                     ELSE                                                            <||||
0139 C                     EXCPTPRSTMT                     PRINT STMT ALONE                 ||||
0140 C                     END                                                  ------------+|||
0141 C**                                                                                     |||
0142 C                     END                                                  -------------+||
0143 C                     END                                                  --------------+|
0144 C                     END                                                  ---------------+
0145 C**----------------------------------
0146 CSR         NXTLVL    BEGSR                           SET COLUMN LEVEL
0147 C**----------------------------------
0148 C** MARK CONTINUATION LINE AND MOVE LEFT ONE COLUMN FOR NEXT NESTED LEVEL
0149 CSR         C         IFGT *ZERO                      LESS THAN 16 LEVELS  ---------------+
0150 CSR                   MOVE '|'       CL,C             MARK CONT. LINE                     |
0151 CSR                   MOVEALVL,C     OUT              FILL OUTPUT AREA                    |
0152 CSR                   END                                                  ---------------+
0153 C**
0154 CSR                   SUB  1         C                MOVE LEFT
0155 CSR         ENDLVL    ENDSR
0156 C**----------------------------------
0157 O**
0158 O** UPDATE CALCS WITH NESTED SYMBOLS IF UPDATE-YES
0159 ONESTWK  E                UPDREC
0160 O                         OUT       96
0161 O**
0162 O** PRINT SOURCE MEMBER WITH NESTED SYMBOLS IF PRINT-YES
0163 OPRINTER E  103           HDNGS
0164 O                         RH,1      85
0165 O                         MEMBER    40
0166 O                         SYSMDYY   57
0167 O                         SYSHM     64 '      '
0168 O                         PAGE      80
0169 O**
0170 O        E  2             HDNGS
0171 O                         RH,2      85
0172 O                         FRMLBR    40
0173 O**
0174 O        E  1             HDNGS
0175 O                         RH,3      85
0176 O**
0177 O        E  1             PRNEST
0178 O                         STMT      96
0179 O                         OUT       96
0180 O**
0181 O        E  1             PRSTMT
0182 O                         STMT      96
0183 O**
0184 O        E  1             SKIP
** RH - HEADINGS FOR PRINTOUT
NESTED LISTING OF RPG PROGRAM - XXXXXXXX         XX/XX/XX  HH:MM        PAGE0000
                 FROM LIBRARY - XXXXXXXX
*    +....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
** LVL - NESTED LEVEL SYMBOLS
+|||||||||||||||
-+||||||||||||||
--+|||||||||||||
---+||||||||||||
----+|||||||||||
-----+||||||||||
------+|||||||||
-------+||||||||
--------+|||||||
---------+||||||
----------+|||||
-----------+||||
------------+|||
-------------+||
--------------+|
---------------+
```

Overhead in External Program Calls

answered by Mel Beckman

Q I use ASNA's RPG/III product to call programs from within other programs. I would like to move field editing from internal subroutines to separately compiled, external programs, but I am concerned about the negative effect this change might have on my system's interactive response time. Is this concern legitimate? If so, do you know of any other way to create a library of commonly used functions so they don't have to be compiled into each program?

A As long as you don't iteratively call an external RPG program (i.e., in a loop), the time required to carry out the call has little effect on interactive response time. The time required to call an external program is usually under 20 milliseconds. For example, let's say a single interactive transaction requires 10 external program calls to do quick calculations, such as date conversions or table lookups. The response time for that transaction would be 200 milliseconds (i.e., two-tenths of a second) slower than if the routines had been coded in-line in the RPG program with EXSR/BEGSR.

On the AS/400, calls take less than one millisecond, so they degrade response time even less than they do on the S/36. External-program-call products that are faster than ASNA's are available if you need them; I believe BPS publishes a CALL time of only one or two milliseconds for repetitive calls to the same module for its RPG 2 1/2 CALL/PARM.

If a given module isn't called for a while, or if your system is badly overloaded, that module may be paged out to disk, in which case a program call takes an additional 30 to 50 milliseconds to page the module into memory (regardless of the size of the module). But small modules called on every transaction most likely remain resident.

Using External Program Calls in COBOL/36

by Lou Forlini

Q Do you know of a product (or method) that lets COBOL programs on the S/36 do external program calls (similar to ASNA's RPG III and BPS' RPG 2 1/2 CALL/PARM capability)? The called programs must exist outside of the calling program's 64 K region, not inside it as IBM S/36 COBOL does, and the number of programs to be called can't be limited to just one.

A IBM has a COBOL Dynamic-Call PRPQ that lets you call, chain, and cancel external COBOL programs (only). The program number is 5799-CFJ for the 5360/62 and 5799-PBJ for the 5363/64. The PRPQ works pretty much as you would expect, except that you need to set up a parameter list that includes the length of the data you are passing.

Using ICF-INTRA to Implement External Program Calls

by George Biernadski

Code on diskette:
Procedures ICMAIN, ICCALL
RPG programs ICMAIN, ICCALL

Well, your company's budget has been completed, and once again, no money was allocated for a programming language with CALL/PARM features for your S/36 — what to do? Try the next best thing: CALL/PARM on the S/36 using the Interactive Communications Feature Intra Subsystem (ICF-INTRA). My step-by-step instructions and code will have you up and calling in no time. With three simple configuration screens, you can create an ICF subsystem capable of doing external program calls. After learning how to set up and enable a subsystem to carry out your instructions, you'll learn how to build workstation programs, how to set up calling and called programs, and how to adapt calling and called programs for your own applications. Finally, you'll see how to use a built-in IBM facility to trace data and commands between your external programs.

The ICF-INTRA facility lets you communicate between programs on the same S/36. Interprogram communications revolve around ICF sessions; a session is a pipeline through which you can send and receive information after establishing communications with another program through a workstation file. You can have up to 260 user-acquired sessions within a program and thus can communicate with more than one external program at a time. For example, you could use one external program to handle date manipulation, another to calculate amortization, and another to format a person's name. Making each of these subroutines an external program eliminates the headache of recompiling all the programs that use these routines when you have to change some code. Because the called programs are loaded at execution time, you need recompile only the external program. And, using this technique, you can build true modular systems that are easy to maintain and enhance. You can use this article's programs for the same functions between two different machines using ICF-PEER or ICF-APPC.

How to Configure ICF-INTRA

To use the ICF-INTRA facility on the S/36, you need IBM's Base Communications feature 6001, Release 5.1 (Release 5.0 contains too many bugs). If you don't have it, you can order it free. After installing feature 6001, you must configure the ICF-INTRA subsystem. You may want to create a separate library for all your ICF-INTRA programs and configurations. I use a library called $S36ICF that contains the ICF configuration and the two sample programs used in this article, ICMAIN and ICCALL. In this example, ICMAIN, the main program, establishes a session, sends and receives data, and controls execution. ICCALL, the called program, contains a simple routine that increments by one the numeric parameter being sent. You can model your own ICF-INTRA models on this simple framework to build useful systems.

To configure the ICF-INTRA subsystem, enter CNFIGICF, and the screen in Figure 16-35a appears. Enter INTRA in the Configuration member name field, $S36ICF (or your own library name) in the Library name field, and select option 1 (Create new member). The next screen (Figure 16-35b) asks you what type of subsystem to configure; enter 1 for INTRA. Finally, the third screen (Figure 16-35c) prompts you for the remote location name; I use INTRA for consistency. The final step is to enable the subsystem to carry out your instructions by using the command

```
ENABLE INTRA,$S36ICF
```

You disable the subsystem when your ICF-INTRA programs are done for the day by using the command

```
DISABLE INTRA
```

Coding ICF-INTRA Workstation Files

With an ICF-INTRA subsystem configured, you're ready for the next step: adding ICF workstation file code to your RPG programs because ICF-INTRA uses workstation files to pass data between two programs. Even if your programs aren't "proper" workstation programs (i.e., use no interactive screen formats), you must still code an F-spec to describe a workstation file in each ICF-INTRA program. Be sure the record length is large enough to accommodate the largest data stream you expect to exchange between the two programs, plus four extra bytes to contain the data stream length. My example programs use a workstation record length of 256 bytes (even though the length of my data stream is considerably less — the extra space makes adding fields easier later on), although you can use any value between four and 4,096.

You also must code a number of F-spec continuation lines — KID, KFMTS, KNUM, KINFDS, and KINFSR — to further describe the workstation file to the INTRA subsystem. Both the calling and the called programs require similar continuation lines; the only difference is the KNUM line, which isn't used in the called program unless the called program is compiled as a multiple requester terminal (MRT) program.

For continuation line KID (Figure 16-36, line 10), use a field name to which you are accustomed for the workstation ID; my program examples use WSID. Workstation IDs contain two characters; the first character is alphabetical, and the second character is alphanumeric. To set up an ICF session, you also need a two-character symbolic session ID; however, the first character is numeric, and the second character is alphabetical. In my examples, 1A is the ICF session ID used in both the procedure of the calling program and its source code. The statement // SESSION LOCATION-INTRA,SYMID-1A sets up an ICF session for communications. If you want several programs to communicate simultaneously, you need a separate session OCL statement with a unique symbolic ID for each program.

In my examples, I use *NONE for continuation line KFMTS (line 11) because this example does not use screen formats; however, if you write an on-line workstation program, you need screen formats and can eliminate this continuation line.

Continuation line KNUM (line 12) tells the program how many devices are attached; my example specifies two, one for the workstation of the calling program and one for the ICF session. If you are calling more than one program, the KNUM value should reflect the number of called programs plus one. Remember, you don't need the KNUM continuation line in the called program.

Continuation line KINFDS (line 13) specifies the workstation INFDS in my examples. If you have written workstation programs, you probably have used the Information Data Structure (INFDS) to check for error conditions or to see whether a function key was pressed. ICF uses the INFDS to send return codes to report on the success of the last attempted operation. The return code is found in positions 23 to 26 of the INFDS and is alphanumeric. These return codes are the same ones displayed on the ICFDEBUG screen under the MAJ/MIN heading.

Return codes are broken into two parts: the first two characters are the major code, and the last two characters are the minor code. Major codes 00 through 03 indicate success, 04 indicates a problem, and 08 through 34 indicate miscellaneous program errors that cause program halts. The minor code further identifies the return code.

Actions for specific codes are handled in the exception processing subroutine, INFSR, as specified on the workstation continuation line. In my examples, KINSFR is INFSR (line 14), although the subroutine can be called anything as long as you uphold naming conventions. Note that my INFSR routines (Figure 16-36, lines 84 through 86) do not contain calculation lines — if a program is operating smoothly and you don't attempt to get too fancy with it, there's no reason to worry about handling exception and error conditions; however, to print the return code and each program cycle, you may want to insert DEBUG statements for use during testing.

Finally, if you want to use IDDU-defined formats, use the KCFILE continuation option with the name of an IDDU file definition — my examples do not use IDDU format.

The Main Calling Program — ICMAIN

Program ICMAIN (see Figure 16-37 for procedure ICMAIN) establishes the ICF session, calls the external program, passes one parameter, receives the processed parameter, terminates the external program, and ends the ICF session. Let's look specifically at how program ICMAIN performs these tasks.

From the F-specs we move to the I-specs of ICMAIN. The external program returns data with a record-identifying code placed in the first two positions of the data stream. The returned data appears as an input field on line 16; any number of input fields can be returned. My example subprogram sends back only one record format (identified by the record type $1). In your programs, each returned record format should be identified by a unique record type. Because of the four bytes containing the data stream length, the beginning and ending positions of the fields are four bytes less than they are on the O-spec-defined data stream. You also must use a dummy record type line (line 17) for the initial workstation read. Lines 18 through 20 define the INFDS that holds the return code for each ICF operation.

Lines 21 through 23 consist of three subroutine calls that set up the communications pipeline by establishing a session, sending and receiving data, and terminating the program.

The first subroutine, ICACQ (lines 31 through 46), performs a dummy workstation read to retrieve the workstation ID, establishes a session with an ACQ operation, and follows with an EXCPT operation that uses the EVOK exception records (lines 87 through 93) to issue the ICF command $$EVOK that starts the called program. Because each ICF session has a unique symbolic ID (1A in my example), you should save the actual workstation ID in case you need it later. Because each session requires logical IDs and because workstation IDs are also logical IDs, 1A is moved into the WSID field to acquire the ICF session. If there is an execution error, the program halts and displays an error message. ICF operation code $$EVOK starts called program ICCALL (see lines 88 to 93 for the evoke parameter list). The NEXT operation code (line 41) forces input from the device described by WSID (1A) and performs a read to ensure that the programs are communicating. As with the initial workstation read, no data is received because no data was sent by program ICCALL. Finally, the WSID field is restored with the real workstation ID (line 44).

Notice that the WSID field is saved and restored constantly; should your program processing also include a display station in addition to an ICF session, you must put the proper device ID into the WSID field before performing workstation output to the physical workstation device. Because the example programs do not use a display station, you could eliminate the save and restore operations on lines 44, 54, 61, and 70.

Subroutine ICCARE (lines 52 through 63) actually sends and receives data. Because you are sending data using the ICF session, workstation ID 1A is moved into the WSID field before EXCPT is issued (see the SEND parameter list, lines 95 through 99). Then, the NEXT operation code, followed by a workstation READ, retrieves the data (which has the $1 record type) sent back through the workstation input field $PARM. After completing this portion of the processing, the program restores the WSID field.

The session termination subroutine, ICTERM (lines 68 through 79), begins by resetting the ICF session ID again. It then uses an EXCPT operation to send an end of transaction (EOT) ICF command to tell the external program to detach and terminate. Finally, it uses an EXCPT operation to send an end-of-session ICF command to terminate the ICF session; should you need it again, you must acquire the session with an ACQ operation.

Let's take a closer look at the various "parameter lists" that appear as exception output in the O-specs. The EVOK parameter list begins with $$EVOK (line 88), which starts up an external procedure that contains the load/run statements to start the program. When using ICF, the screen format name becomes the ICF command name. Byte assignments following use of the $$EVOK command are:

1 - 8: name of program (procedure) to activate

9 - 16: password (if security is active)

17 - 24: user ID (it's a good idea to always use one)

25 - 32: library of program (or procedure) to activate

33 - 52: blank

53 - 56: 0000(data stream length of zero)

Should you want to send data when you start the called program, set bytes 53 through 56 to equal the data stream length. Bytes 57 and up can contain output data that will be received during the initial workstation read of the called program; you must set the program data and the include statements on the called procedure to yes (Y).

The output data stream format (lines 95 through 99) begins with a $$SEND ICF command; a maximum of 4,092 bytes is allowed. Bytes 1 through 4 indicate the total length of the output data stream (not including these four bytes); bytes 5 through *N* (where *N* is any number equal to or less than 4,096) contain the output data stream. Remember that the ending positions of the O-spec data stream are four higher than the corresponding I-spec positions; the first four bytes (i.e., the data stream length) are not received as data by the called program.

The SENDET parameter list operation (lines 101 through 103) ends the link to the external program. The EOT sends code 0308 to the INFDS in the called program where INFSR processes the code and terminates the program. Unless you are sending data with the $$SENDET operation code, place 0000 in bytes 1 through 4 for the output data stream length.

Finally, the EOS parameter list (line 105) ends the ICF session. The ICF command is $$EOS.

Called Program — ICCALL

Program ICCALL (Figure 16-38), the externally called program, is much simpler and less involved than program ICMAIN (see Figure 16-39 for procedure

ICCALL). Program ICMAIN handles much of the overhead; consequently, program ICCALL is relatively simple to code. In addition, you don't have to worry about acquiring a session or saving and restoring the workstation ID. Finally, if you will be receiving data with the initial workstation read, remember to save procedure ICCALL with the Include Program Data attribute set to yes (Y). As with the main program, a line-by-line description follows.

Lines 15 through 17 contain input code in addition to a dummy record ID line for the initial workstation read. Notice that ICMAIN O-specs define the record ID ($1), ending in position 6, but program ICCALL receives the record ID, ending in position 2. This difference occurs because of the four-byte data stream length, which isn't transmitted as data to the called program.

The INFDS (lines 18 through 22) contains the return code generated by ICF and is broken into two parts; the first two positions are the major return code, and the last two positions are the minor return code. Your concern is with the minor return code; when it has a value of 08, you know the $$SENDET ICF operation code was sent and that you should terminate the program.

Lines 23 through 32 contain processing code. When data is actually received, the subroutine processes the particular record type. Any processing can go here; for example, you could manipulate dates, calculate amortization, or, in the case of my example, increment the received parameter by one.

The INFSR subroutine (lines 37 through 41) checks for a minor return code of 08, which signals you to terminate the program. The *DETC used as Factor 2 of the ENDSR operation tells the program to resume execution at line 23 (the beginning of detail calculations) after finishing the INFSR subroutine.

Code for the output data stream is found in lines 42 through 49. The first output record acknowledges to ICMAIN that the called program is indeed running by sending a data stream of one blank. This record is processed by the READ operation on line 42 of program ICMAIN. The second output record contains the processed data to be sent back to the calling program. The output ending positions are four higher than the input position in program ICMAIN, due once again to the output data stream length in output positions 1 through 4. This record is processed by the READ statement on line 59 of program ICMAIN.

Rules for Program Modification

To adapt ICMAIN and ICCALL for your own applications, you must change the program name, the input and output data streams, and the actual information processing logic. In these places, you can add and change as much code as you want — everything else should remain consistent from program to program. As an alternative to having many programs, each with its own routine, you can group routines into one or a few programs. Use different record types so you can distinguish easily between routines you are calling within the same program.

Use of ICFDEBUG

ICFDEBUG is analagous to a trace table, a handy utility, supplied by IBM for tracing data and commands sent to and from ICF programs. Enter ICFDEBUG ON to activate ICFDEBUG, and enter ICFDEBUG OFF to deactivate it. Reset or blank out by entering ICFDEBUG ON. You can view the trace table by entering ICFDEBUG CRT before entering ICFDEBUG OFF. Figure 16-40 shows what a typical page looks like; in fact, it traces the I/O of my sample programs. Figure 16-40 also includes explanations of the headings.

Now you have the information you need to install ICF-INTRA to call external programs — can you think of anything else your S/36 might receive that's free and as useful? ICF in itself is interesting, and it's beneficial in instances where modular code should be used; I hope this article has illuminated some of its facets.

For additional information on ICF, you can refer to *Interactive Communications Feature* (SC21-9533-0) and the Interactive *Communications Feature: Guide and Examples* (SC21-7911-3).

Figure 16-35a
Configuration member definition

```
 1.0          SSP-ICF CONFIGURATION MEMBER DEFINITION

1  Configuration member name                                      INTRA

2  Library name   . . .                                           $S36ICF

3  Select one of the following·
      1. Create new member
      2. Edit existing member
      3. Create new member from existing member
      4. Remove a member
      5. Review a member
   Option . . .           .                                  1-5  1



Cmd7-End      Cmd19-Cancel

 Help text available throughout CNFIGICF by pressing the Help key
```

Figure 16-35b

Configuration member type

```
 2 0                 SSP-ICF CONFIGURATION MEMBER TYPE            INTRA

Select one of the following options
      1  Intra
      2  BSC
      3  SNA
      4  Async
      5. PC Support/36

   Option  1

Cmd3-Previous display        Cmd5-Restart CNFIGICF
Cmd7-End                     Cmd19-Cancel                 COPR IBM Corp. 1986
```

Figure 16-35c

Subsystem member definition

```
 22 0                 SUBSYSTEM MEMBER DEFINITION                INTRA
W1

 1  Remote location name          . . .                          INTRA

Cmd5-Restart CNFIGICF            Cmd7-End
Cmd19-Cancel                                              COPR IBM Corp  1986
```

Figure 16-36

Program ICMAIN

```
*        1 .       2         3         4  . .  5 .       6         7     . 8
H    P 64                     B       1                              ICMAIN
F*
F*    NAME- ICMAIN
F*    DATE-  6/01/88
F*  AUTHOR- GEORGE A BIERNAOSKI. COPYRIGHT (C) 1988
F*
F*  FUNCTION- ICF MAIN PROGRAM
F*
FWSICMAINCD  F     256            WORKSTN
F                                              KID     WSID
```

```
F                                                 KFMTS  *NONE
F                                                 KNUM        2
F                                                 KINFDS INFDS
F                                                 KINFSR INFSR
IWSICMAINWS      1 C$   2 C1                                      * RETURN RECORD TYPE
I                                       3   70$PARM               * PROCESS DATA
I           WS                                                    * INITIAL READ
IINFDS         DS                                                 * RETURN CODE STRUC
I                                                                 * STATUS  STATUS
I                                      23  26 RECODE              * MAJOR CODE
C                    EXSR ICACQ                           * INITIAL PROCESSING
C                    EXSR ICCARE                          * MAIN PROCESSING
C                    EXSR ICTERM                          * TERMINATE PROCESSING
C*
C****************************************
C* ICACQ  - INITIAL PROCESSING          *
C*                                      *
C*        - ACQUIRE ICF SESSION         *
C*        - CALL EXTERNAL PROGRAM       *
C****************************************
CSR          ICACQ     BEGSR
C*
C                    READ WSICMAIN                        * INITIAL READ
C                    MOVELWSID      SAVEID  2             * SAVE WORKSTATION ID
C                    MOVEL'1A'      WSID                  * SESSION SYMBOLIC ID
C*
C            WSID    ACQ  WSICMAIN                        * ACQUIRE SESSION
C*
C                    EXCPTEVOK                            * '$$EVOK   ' START PROGRAM
C*
C            WSID    NEXT WSICMAIN                        * GET NEXT INPUT FROM SESSION
C                    READ WSICMAIN                        * READ SESSION/ACKNOWLEDGE START
C*
C                    MOVELSAVEID    WSID                  * RESTORE SAVE ID
C*
C                    ENDSR
C*
C****************************************
C* ICCARE - CALL PROGRAM WITH DATA      *
C*        - RECEIVE PROCESSED DATA      *
C****************************************
CSR          ICCARE    BEGSR
C*
C                    MOVEL'1A'      WSID
C*
C                    EXCPTSEND                            * '$$SEND   ' SEND DATA
C*
C            WSID    NEXT WSICMAIN                        * GET NEXT INPUT FROM SESSION
C                    READ WSICMAIN                        * READ SESSION
C*
C                    MOVELSAVEID    WSID
C*
C                    ENDSR
C*
C****************************************
C* ICTERM - TERMINATE PROCESSING        *
C****************************************
CSR          ICTERM    BEGSR
C*
C                    MOVEL'1A'      WSID
C*
C                    EXCPTSENDET                          * '$$SENDET' END OF TRANSACTION
C*
C                    EXCPTEOS                             * '$$EOS    ' END OF SESSION
C*
C                    MOVELSAVEID    WSID                  * RESTORE WORKSTATION ID
C                    SETON                     LR
C*
C                    ENDSR
C*
C****************************************
C* INFSR  - RETURN CODE PROCESSING      *
C****************************************
CSR          INFSR     BEGSR
C*
```

```
C                    ENDSR
OWSICMAINE                EVOK                                    * CALL PROGRAM
O                                 K8 '$$EVOK  '
O                                  8 'ICCALL  '                   * PROGRAM NAME
O                                 16 '        '                   * PASSWORD
O                                 24 'USERID  '                   * USER ID
O                                 32 '$S36ICF '                   * PROGRAM LIBRARY
O                                 56 '0000'
O*
O        E                SEND                                    * SEND DATA
O                                 K8 '$$SEND  '
O                                  4 '0007'
O                                  6 '$1'                         * RECORD TYPE
O                                 11 '00001'                      * DATA
O*
O        E                SENDET                                  * TERMINATE CALL PROG
O                                 K8 '$$SENDET'
O                                  4 '0000'
O        E                EOS                                     * TERMINATE SESSION
O                                 K8 '$$EOS   '
```

Figure 16-37

Calling procedure ICMAIN

```
// LOAD ICMAIN
// SESSION LOCATION-INTRA,SYMID-1A
// RUN
```

Figure 16-38

Program ICCALL

```
  . . . 1 . . . . 2 . . . . 3 . . . . 4 . . . . 5 . . . . 6 . . . . 7 . . . . 8
H    P 64                          B       1                           ICCALL
F*
F*     NAME- ICCALL
F*     DATE-  6/01/88
F*   AUTHOR- GEORGE A BIERNADSKI, COPYRIGHT (C) 1988
F*
F*   FUNCTION- ICF CALL PROGRAM
F*
FWSICCALLCP  F     256             WORKSTN
F                                                KID    WSID
F                                                KFMTS  *NONE
F                                                KINFDS INFDS
F                                                KINFSR INFSR
IWSICCALLWS        1 C                                                * INITIAL READ
I         WS  01   1 C$    2 C1                                       * DATA RECORD TYPE
I                                          1   2 $RECID               * RECORD TYPE
I                                          3   70$PARM                * INPUT DATA
IINFDS       DS                                                       * RETURN CODE STRUC
I                                        *STATUS  STATUS
I                                         23  26 RECODE               * RETURN CODE
I                                         23  24 MACODE               * MAJOR CODE
I                                         25  26 MICODE               * MINOR CODE
C   01NLR             EXSR SUBR01                         * PROCESS RECORD
C*
C****************************************
C* SUBR01 - PROCESS '$1' RECORD TYPE      *
C****************************************
CSR         SUBR01    BEGSR
C*
C                     ADD  1         $PARM                * PROCESS DATA
C*
C                     ENDSR
```

```
0033 C*
0034 C****************************************
0035 C* INFSR  - RETURN CODE PROCESSING       *
0036 C****************************************
0037 CSR         INFSR     BEGSR
0038 C*
0039 C           MICODE    COMP '08'                     LR* END OF TRANSACTION
0040 C*
0041 C                     ENDSR'*DETC'
0042 OWSICCALLD       NO1NLR                                          * ACKNOWLEDGE START
0043 O                              K8 '$$SEND  '
0044 O                               4 '0001'
0045 O         D        O1NLR                                         * SEND BACK DATA
0046 O                              K8 '$$SEND  '
0047 O                               4 '0007'
0048 O                     $RECID    6                                * RECORD TYPE
0049 O                     $PARM    11                                * PROCESSED DATA
```

Figure 16-39

Calling procedure ICCALL

```
// LOAD ICCALL
// RUN
```

Figure 16-40

Facility ICFDEBUG and heading explanations

```
JOB NAME PROC NAME PROG NAME LOC NAME FORMAT NAME SYM ID  MAJ/MIN   OPERATION CODE DATA LENGTH  DATA -
W1143415 ICMAIN    ICMAIN    INTRA                1A/      0000          ACQ
W1143415 ICMAIN    ICMAIN    INTRA      $$EVOK    1A/      0001          EVI            0000
01143420 ICCALL    ICCALL    INTRA      $$SEND    01/1A    0101          PTI            0001
W1143415 ICMAIN    ICMAIN    INTRA                1A/01    0000          GET            0001
W1143415 ICMAIN    ICMAIN    INTRA      $$SEND    1A/01    0001          PTI            0007      $100001
01143420 ICCALL    ICCALL    INTRA                01/1A    0000          ACI            0007      $100001
01143420 ICCALL    ICCALL    INTRA      $$SEND    01/1A    0001          PTI            0007      $100002
W1143415 ICMAIN    ICMAIN    INTRA                1A/01    0000          GET            0007      $100002
W1143415 ICMAIN    ICMAIN    INTRA      $$SENDET  1A/      0000          PEX            0000
01143420 ICCALL    ICCALL    INTRA                01/      0308          ACI            0000
W1143415 ICMAIN    ICMAIN               $$EOS     1A       0000          EOS

JOB NAME - same as on 'status users' console command
PROC NAME - procedure that is active
PROG NAME - program that generated the operation and data
LOC NAME - ICF-INTRA system being used
FORMAT NAME - ICF operation code
SYM ID - session id (source/target)
MAJ/MIN - return code found in INFDS data structure after each operation
OP CODE - operation code that is performed
ACQ - acquire
EVI - evoke then invite
PTI - put the invite
ACI - accept input
GET - RPG 'READ' operation
PEX - end of transaction
EOS - end of session
DATA - data stream that was transmitted (excluding the first four bytes which are the header)
```

Using Dynamically Privileged RPG Subroutines

by Mel Beckman and Bob Schuette

Code on diskette:
Assembler subroutines SUBRDP, SUBRNP

Q We recently purchased several S/36 assembly language subroutines from two different vendors. The routines work fine — as long as we use them in separate programs. When we try to use routines from both vendors in the same program, however, the program halts with the system message, "Privileged Operation Attempted In Nonprivileged Mode." One vendor explained that his routines run "dynamically privileged," making them incompatible with subroutines that run "continuously privileged." We really want to combine the power of both vendors' routines in a single program. Is the vendor's explanation valid? Can anything be done to make the two vendors' products compatible?

A Your vendor's explanation is valid. The S/36 supports two "privileged" modes, continuous and dynamic. In continuously privileged mode, the entire program can always access IBM's privileged machine instructions. This is a somewhat dangerous situation because bug-ridden code that inadvertently executes a privileged instruction can crash the entire system. To reduce this danger, a program can use dynamically privileged mode and access privileged instructions only when needed. In dynamically privileged mode, there is consequently much less chance that the program will be privileged when a bug is encountered. The bug then has much less effect on the system.

If you mix continuously and dynamically privileged subroutines in the same program, a problem arises. When the dynamically privileged subroutine is called, it turns off privileged mode before returning to the main program. When the main program subsequently calls a continuously privileged subroutine and privileged mode has been turned off, the subroutine fails when it attempts a privileged operation, yielding the message you receive.

The solution is to turn privileged mode back on just before calling the continuously privileged subroutine. You should also turn privileged mode off again to retain the protection provided by dynamically privileged mode. Two assembler subroutines, SUBRDP and SUBRNP, turn privileged mode on and off respectively.

You should call SUBRDP just before calling the continuously privileged routine and then call SUBRNP immediately afterward. (See Figure 16-41 for an example of using the two subroutines.) SUBRDP and SUBRNP can be used in programs that don't call dynamically privileged subroutines to obtain the protection of dynamically privileged mode with continuously privileged subroutines.

Figure 16-41

Example of using SUBRDP and SUBRNP

```
C                       EXIT SUBRDP                     Prtx mode on
C*
C                       EXIT SUBR[illegible]            Call sort
C                       RLABL          STRIAL  8        printed routine
C*
C                       EXIT SUBRNP                     Prtx mode off
```

Re-creating Subroutine SUBRDP

If you don't have assembler subroutine SUBRDP, you can re-create it with procedure MKSUBRDP (you don't need IBM's Assembler Language Program Product to install SUBRDP). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRDP. You need to run MKSUBRDP only once to create the SUBRDP subroutine.

```
// * 'Re-creating R-module SUBRDP in library #RPGLIB
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'0000005B'     Number of #MAINT records
// LOCAL OFFSET-208,DATA-+
'D9E2E4C2D9C4074040000002000000000000080400000008900012000000000'88'
// LOCAL OFFSET-273,DATA-+
'0920155631000000803D8880000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRDP member to insert object code
// LOAD $FEFIX
// RUN
HDR  3B50 SUBR000000
PTF  1B6F RSUBRDP,99,,#RPGLIB
DATA B658 00 0000 E2D8E2E4C2D9C407[illegible]
DATA 4019 00 0020 [illegible]
DATA 0AAC 00 0040 E3050005F4000AF087[illegible]
DATA B370 00 0060 [illegible]
DATA 2168 00 0080 C5FFFFF0[illegible]
DATA 202E 00 00A0 [illegible]
DATA FA80 00 00C0 615C[illegible]
DATA 1B44 00 00E0 [illegible]
END  CC98
```

Re-creating Subroutine SUBRNP

If you don't have assembler subroutine SUBRNP, you can re-create it with procedure MKSUBRNP (you don't need IBM's Assembler Language Program Product to install SUBRNP). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRNP. You need to run MKSUBRNP only once to create the SUBRNP subroutine.

Continued

```
// * 'Re-creating R-module SUBRNP in library #RPGLIB '
* Build an empty member in a $MAINT fila with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000039'    Number of $MAINT records
// LCCAL OFFSET-209,DATA-+
'09E2E4C2D905D7404000000020000000000000604000000009900012000000000389'
// LOCAL OFFSET-273,DATA-+
'0920155631000000B03D3300000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLDCKS-25,EXTENO-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRNP member to insert object code
// LOAD $FEFIX
// RUN
HDR  3B94 SUBRN00000
PTF  A3B4 RSUBRNP,99,,#RPGLIB
DATA 7EBC 00 0000 E20BE2E4C2D9D5D7000000001A00000000000000000000000000000000000000
DATA 0850 00 0020 000000C000000000000000000000000000000000000000000000000000000000
DATA 4976 00 0040 E316001634A10019F4000F8A25007A103035A10019F68001F087000000000000
DATA 1345 00 0060 0000000000000000000000000000000000000000000000000000000000001003
DATA A183 00 0080 C5FFFF1600000000000000000000000000000000000000000000000000000000
DATA A146 00 00A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 7C9A 00 00C0 615C000000000000000000000000000000000000000000000000000000000000
DATA 9B5E 00 00E0 0000000000000000000000000000000000000000000000000000000000000000
END  4DB5
```

Using RPG Assembly Language Subroutines in COBOL Programs

by Mel Beckman

Code on diskette:
Assembler subroutine RBRIDG
COBOL program TBRIDG

Use assembly language subroutine RBRIDG to let your COBOL programs access RPG assembly language subroutines.

At last count, more than 230 assembly language subroutines existed in the S/36 marketplace. These routines provide access to machine and operating system capabilities not directly accessible through high-level languages (HLLs), frequently making the impossible possible for many applications.

Alas, if your HLL of choice is COBOL, you're constrained to using a mere two dozen of these technical gems because most of the routines interface with RPG only. The inscrutable IBM chose long ago to use different assembler subroutine linkage conventions for RPG and COBOL, making each camp's routines inaccessible to the other.

Until now. Assembly language subroutine RBRIDG lets you build a bridge between your COBOL program and most, if not all, existing RPG assembler routines. Using RBRIDG is simply a matter of defining, in your COBOL program's WORKING-STORAGE section, the RLABL parameters expected by any RPG assembler subroutines you want to use. Then, before calling the routine itself, you just make a call to RBRIDG to build a bridge to the desired routine.

Defining RLABLs

Subroutine RBRIDG interfaces with any assembler routine that you can call via the RPG EXIT operation, as long as the routine doesn't require indicator or array parameters. (COBOL has no RPG-like indicator area or array definitions.) To use subroutine RBRIDG in a COBOL program, you must first build, in the WORKING-STORAGE section, an RLABL definition list for each subroutine you plan to call (see Figure 16-42 for an example of coding an RLABL definition list). In the 01-level data description entry, code a name for the definition list; later, you'll pass this name to RBRIDG. Each RLABL the target subroutine uses has a corresponding RLABL definition within this 01-level item. Each definition consists of three data items: type, length, and the data field itself. The type item is a one-character variable containing F for RPG field RLABLs and D for RPG data structure RLABLs. The length item is a two-byte COMP-4 (binary) variable containing the length of the RLABL field. The data field item represents the RPG field or data structure — it contains data being exchanged with the target subroutine — and is the only data description item you must name uniquely. All other items can have the name FILLER.

You can code as many RLABL definition entries as you like. After the last entry, code a one-byte FILLER with a value of E to mark the end of the definition list.

Making Your Call

With an RLABL definition list, using RBRIDG to call an RPG assembler subroutine is simple (see Figure 16-43 for an example). Just code a CALL to subroutine RBRIDG, specifying the name of the RLABL definition list in the USING clause. Immediately follow this CALL with a CALL to the target subroutine, without a USING clause. Note that you can't code any statements between the two CALLs. If subroutine RBRIDG detects a statement between the two CALL statements or an error in the RLABL definition list (e.g., the length item doesn't match the actual data field length), it halts with an error message. Figure 16-44 gives a sample COBOL program that calls the RPG assembler subroutine SUBRLD to read a library directory and print it.

Now that you've got a bridge-making tool, you can start crossing the river to all those great RPG assembler routines you've done without for so long.

Figure 16-42

Example of coding an RLABL definition list

```
 WORKING-STORAGE SECTION
*
* RLABL definition list for three RLABLs
*       RLABL        FIELD1       A ten-byte field
*       RLABL        FIELD2       A one-byte field
*       RLABL        DSTRUC       A 300-byte data structure
*
 01  SUBRXX-RLABLS
     05  FILLER
         10  FILLER               PIC A         VALUE 'F'
         10  FILLER               PIC 9999 COMP-4 VALUE 10
```

```
    10  SUBRXX-FIELD1        PIC A(10)
05  FILLER
    10  FILLER               PIC A           VALUE  F'
    10  FILLER               PIC 9999 COMP-4 VALUE 1
    10  SUBRXX-FIELD2        PIC A(1)
05  FILLER
    10  FILLER               PIC A           VALUE 'D'
    10  FILLER               PIC 9999 COMP-4 VALUE 300
    10  SUBRXX-DSTRUC        PIC A(300)
05  FILLER                   PIC A           VALUE 'E'
```

Figure 16-43

Example of coding CALL statements for RBRIDG

```
CALL 'RBRIDG' USING SUBRXX-RLABLS
CALL 'SUBRXX'.
```

Figure 16-44

Sample COBOL program using RBRIDG

```
************************************************************
*                                                          *
*         This is a sample COBOL program that tests the    *
*         RBRIDG (RPG Assembler Subroutine Bridge)         *
*                                                          *
************************************************************
PROCESS MAP,OFFSET
IDENTIFICATION DIVISION
PROGRAM-ID       TBRIDG
AUTHOR           MEL BECKMAN
INSTALLATION     BECKMAN SOFTWARE ENGINEERING
DATE-WRITTEN     22 FEBRUARY 1990
SECURITY         NONE
ENVIRONMENT DIVISION
CONFIGURATION SECTION
SOURCE-COMPUTER   IBM-S36
OBJECT-COMPUTER   IBM-S36
SPECIAL-NAMES
    SYSTEM-CONSOLE IS CONSOLE
DATA DIVISION
WORKING-STORAGE SECTION
*
* SUBRLD RLABL parameters
*
*       RLABL          LIBNAM    8        Input
*       RLABL          MEMNAM    8        Input
*       RLABL          MEMTYP    1        Input
*       RLABL          DIRDS    80        Output
*       RLABL          RCODE     1        Output
*
01  SUBRLD-RLABLS
    05  FILLER
        10  FILLER               PIC A           VALUE 'F'
        10  FILLER               PIC 9999 COMP-4 VALUE 8
        10  SUBRLD-LIBNAM        PIC A(8)
    05  FILLER
        10  FILLER               PIC A           VALUE 'F'
        10  FILLER               PIC 9999 COMP-4 VALUE 8
        10  SUBRLD-MEMNAM        PIC A(8)
    05  FILLER
        10  FILLER               PIC A           VALUE 'F'
        10  FILLER               PIC 9999 COMP-4 VALUE 1
        10  SUBRLD-MEMTYP        PIC A(1)
    05  FILLER
        10  FILLER               PIC A           VALUE 'D'
        10  FILLER               PIC 9999 COMP-4 VALUE 80
        10  SUBRLD-DIRDS         PIC A(80)
    05  FILLER
        10  FILLER               PIC A           VALUE 'F'
        10  FILLER               PIC 9999 COMP-4 VALUE 1
```

```
        10  SUBRLD-RCODE          PIC A(1)
     05  FILLER                   PIC A          VALUE 'E'
 PROCEDURE DIVISION
*
* Print all source member directory entries
*
 MAINLINE
     MOVE 'NEWS3438' TO SUBRLD-LIBNAM
     MOVE '        ' TO SUBRLD-MEMNAM
     MOVE 'S       ' TO SUBRLD-MEMTYP
     MOVE '0' TO SUBRLD-RCODE
*
     PERFORM PRINT-DIR-ENTRY
         UNTIL SUBRLD-RCODE IS NOT EQUAL TO '0'
*
* Get out of Dodge
*
 EXIT-PROGRAM
     DISPLAY '*** Test of TBRIDG completed ***'
     STOP RUN
*
* Print a directory entry
*
 PRINT-DIR-ENTRY
     CALL 'RBRIDG' USING SUBRLD-RLABLS
     CALL 'SUBRLD'
     DISPLAY SUBRLD-DIRDS
```

Re-creating Subroutine RBRIDG

If you don't have assembler subroutine RBRIDG, you can re-create it with procedure MKRBRIDG (you don't need IBM's Assembler Language Program Product to install RBRIDG). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKRBRIDG. You need to run MKRBRIDG only once to create the RBRIDG subroutine.

```
// * 'Re-creating R-module RBRIDG in library #LIBRARY'
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-,
'D909C2D9C9C4C74040000000400000000000000600000000009900022000000001890'
// LOCAL OFFSET-273,DATA-,
'07061318310000000000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#LIBRARY
// END
* Patch the new RBRIDG member to insert object code
// LOAD $FEFIX
// RUN
HDR  D384 RBRID00000
PTF  4113 RRBRIDG,99,,#LIBRARY
DATA CDE5 00 0000 E20BD9C2D9C9C4C7000000032A00000000000000000000000000000000000000
DATA 69AD 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA D9BA 00 0040 E3340034F28720D9C2D9C9C4C740F14BF04082A84004859340C2858392948195
DATA 2F6E 00 0060 40404040404040340103253402032734080329340802C30E01000000322E2A28
DATA B846 00 0080 E333006802C303173501032911C01008F0E4D010C0319C00102C4750201B50203
DATA 0E3E 00 00A0 C20100C05F0707078DC500F281518DC600F2011F8C000002001F151108070301
DATA 9213 00 00C0 E336009F7F01007402024E010203136E010202860202E20203020103F1872F8D
```

Continued

```
DATA C03B 00 00E0 C400F201224C0202031C8C0104024F0104031174020B4E0106031500382E250A
DATA EF1C 00 0100 E31F008F8B0202E202030201 07F18757C08702C44C03030088F28704C08702C0
DATA 0E76 00 0120 C0870000000000000000000000000000000000000000000000000000001B140F
DATA 0C58 00 0140 E33B02F8C0870000C2020200F4010405F40104045B0000000A09C209C9000103
DATA 6439 00 0160 208002F00000000000000000000000000000000009C209C9C4C740A681001F07
DATA 8840 00 0180 E32A0323A2408381939386884408995839699998583A393A840404000001000200
DATA 136C 00 01A0 030016C06FFFFFFF3009C209C8C4C70000000000000000000000000000000000
DATA 18E8 00 01C0 C5FFFF2300000000000000000000000000000000000000000000000000000000
DATA 3888 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
END  E910
```

Retrieving the DTF Control Block in COBOL Programs

by Bob French

Code on diskette:
COBOL programs GTDTF1, GTDTF2

The S/36 COBOL compiler includes a neat and easy method for retrieving the Define the File (DTF) control block for a given file. DTFs provide the interface between programs and the SSP's data management support and contain useful information, such as the relative record number of the last record processed or the cursor position for a workstation file. It is similar to the file information feedback area on the S/38 and AS/400.

To retrieve the DTF for a file, first define a WORKING-STORAGE data structure (DTF-LIST) 160 bytes long (Figure 16-45). Next, code a CALL statement to a separate COBOL subroutine (Figure 16-46), passing the name of the file and the DTF-LIST structure. The COBOL subroutine receives as its first parameter the DTF for the file name you passed to it. It then moves this DTF to the second parameter and returns. Your calling program resumes control with the requested file's DTF control block in DTF-LIST. As a result, you can redefine selected subfields within DTF-LIST to retrieve information of interest.

The example shows how to extract the cursor position (row and column) from a workstation file. The cursor row and column numbers are stored as two one-byte binary values in positions 55 and 56 of the DTF. Moving these values individually to COMP-4 variables converts them from binary to decimal. For a description of the DTF control block for any type of file, see the *S/36 System Data Areas Manual* (LY21-0592).

Figure 16-45

Code to retrieve the DTF control block. (This code is contained in member GTDTF1 on diskette.)

```
*
*  Source code required for calling program
*
 WORKING-STORAGE SECTION
*
```

```
*    DTF list contains system data area values based on the file
*
    01  DTF-LIST
      05  FILLER                 PIC X(54)
      05  WS-ROW                 PIC X
      05  WS-COL                 PIC X
      05  FILLER                 PIC X(104)

    01  CONVERT-TO-DECIMAL
      05  ROW-COL1               PIC XX
      05  ROW-COL2 REDEFINES ROW-COL1
        10  FILLER               PIC X
        10  ROW-COL3             PIC X
      05  ROW-COL4 REDEFINES ROW-COL1
        10  ROW-COL              PIC S99        COMP-4

    01  ROW-COLUMN
      05  ROW                    PIC 99
      05  COLUMN                 PIC 99

 PROCEDURE DIVISION
     READ SCREEN-FILE
     CALL 'GETDTF' USING SCREEN-FILE, DTF-LIST
*    Convert Row & Column from binary to zoned decimal
     MOVE LOW-VALUES TO ROW-COL1
     MOVE WS-ROW TO ROW-COL3
     MOVE ROW-COL TO ROW
     MOVE LOW-VALUES TO ROW-COL1
     MOVE WS-COL TO ROW-COL3
     MOVE ROW-COL TO COLUMN
```

Figure 16-46

Subroutine GETDTF to retrieve a DTF. (This code is contained in member GTDTF2 on diskette.)

```
*
*  Source code required for called program (subroutine)
*            Program Name - GETDTF

  LINKAGE SECTION
    01  DTF-AREA                 PIC X(160)
    01  DTF-RETURN-AREA          PIC X(160)


 PROCEDURE DIVISION USING DTF-AREA, DTF-RETURN-AREA
 000-MAINLINE-CONTROL
     MOVE DTF-AREA TO DTF-RETURN-AREA
     EXIT PROGRAM
```

Searching for Strings

by Gary T. Kratzer and Mel Beckman

Code on diskette:

Assembler subroutine SUBR$F

Most methods of string handling in RPG leave much to be desired. With RPG's lack of varied data types and the manipulation capabilities found in most other languages, RPG programmers usually resort to the only sensible method available: arrays. And although RPG arrays are fairly convenient to use, in terms of performance, they are hopelessly slow. Whenever you reference an RPG array with a variable subscript (e.g., ARR,X), hundreds of machine instructions may have to be executed, which dramatically increases a program's overall execution time.

In this article, we focus our attention on string handling problems by providing assembler subroutines to perform common string operations that we, as programmers, face nearly everyday. Don't hesitate to implement these assembler subroutines just because compatibility with other machines (e.g., the AS/400) may be an issue; you can easily rewrite these routines in any language because nothing about them is "smoke and mirrors." The first subroutine we present is SUBR$F, which performs a high-speed string search on fields up to 256 bytes long.

To use subroutine SUBR$F in an RPG program, you must code an EXIT SUBR$F operation, which must be followed by six RLABL statements, a detailed description of which follows:

```
C                          EXIT SUBR$F
C                          RLABL          FUNC     1
C                          RLABL          RESLT   30
C                          RLABL          ARGMNT
C                          RLABL          TARGET
C                          RLABL          LEFTP   30
C                          RLABL          RIGHTP  30
```

• FUNC — a one-byte field that contains a code indicating the type of search you want to perform. An I means "initial search"; use this code every time you want to change the search arguments. An R means "repeat previous search"; use this code to repeat the search using the same arguments you used previously but with different data in the target field. A repeat search is much faster than an initial search because all the initialization code in SUBR$F is not executed.

• RESLT — a three-digit field that will contain the leftmost position of the search string in the target field if a match is found, zero if the string is not found, and negative 1 if you made a coding error in the search parameters (e.g., ARGMNT larger than TARGET, LEFTP greater than RIGHTP).

• ARGMNT — a field (a data structure is not allowed) up to 256 bytes long that contains the search argument. The argument ends with the first blank character unless you enclose the entire argument in single quotation marks. For example, to search for the string NOW IS, which contains an embedded blank, you would pass 'NOW IS' in the argument field. If you enclose the argument in double quotation marks, both upper- and lowercase characters in the target string will match. Thus, if the argument field contains "NOW IS", subroutine SUBR$F will find a match with Now Is, now is, or any other combination of upper/lowercase. In this kind of search, the argument characters must be all uppercase.

• TARGET — a field (a data structure is not allowed) up to 256 bytes long that contains the characters to search through.

• LEFTP and RIGHTP — three-digit fields that specify the leftmost and rightmost margin positions that will restrict the search in the target string. If LEFTP is zero, the value 1 is assumed. If RIGHTP is zero, the search string must start at position LEFTP in the target string to match the argument; this "anchored" search is much faster than a general search because only one compare needs to be performed rather than testing all possible positions.

Using subroutine SUBR$F can greatly increase program performance where string searches are used. A common program function in which subroutine SUBR$F would be useful is sequentially reading a disk file and searching for a given substring in certain "free form" fields, such as names or addresses, within each record.

Re-creating Subroutine SUBR$F

If you don't have assembler subroutine SUBR$F, you can re-create it with procedure MKSUBR$F (you don't need IBM's Assembler Language Program Product to install SUBR$F). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBR$F. You need to run MKSUBR$F only once to create the SUBR$F subroutine.

```
// * 'Re creating R-module SUBR$F in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET 201,DATA '00000135     Number of $MAINT records
// LOCAL OFFSET-209,DATA-
'D9C2E4C2D95BC6404000000B0000000000006000000000099000420000003889'
// LOCAL OFFSET-273,DATA -
'11160957310000008056440000D00000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN J,BLOCKS 25,EXTEND 25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT RETAIN S
// RUN
// COPY FROM DISK,FILE $MAINT,RETAIN R TO-#RPGLIB
// END
* Patch the new SUBR$F member to insert object code
// LOAD $FEFIX
// RUN
HDR  3BAA SUBR$00000
PTF  CEB8 RSUBR$F,99,,#RPGLIB
DATA 3844 00 0000 E20BE2E4C2D95BC60014E602620000000000000000000000000000000000000000
DATA C308 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 0CEE 00 0040 E32D1513340B171E3401171634021 71A3501171E750202B0D900C08116250F01
DATA 46EA 00 0060 1735173 50F01173B173B07021747174 71C000000002B2925231F1D190F0B0703
DATA 7C6D 00 0080 E32E1542173505 1C01173908 1C01173708 0F01173717351C00173B091C01173F
DATA D313 00 00A0 0B1C01173 00B0F01173D173B00001735173BC0002D2B2725201B161210080601
DATA B070 00 00C0 E3321575841 6F875020E7501119D020000C08416FB3501173 97D4000F2010F37
DATA 6BC7 00 00E0 0117203F011735F10211C0B716F83C4017413502173790003 12D29221E141002
DATA 11CC 00 0100 E33215A8000000F201283002173 5F2B2217D7D00F2B1067D7F00F201151C0017
DATA 4B09 00 0120 41000E01173717200F0117391 7203F021735350117 1E750031 2D2927232 11C09
```

Continued

```
DATA 22DD 00 0140 E32E15D7020E2C021733000D0217331726F284060C02173317290C0217441733
DATA A182 00 0160 072017331723F204160E01173D17200F01173B002E2A28211F181916130C0A05
DATA 74F4 00 0180 E32F16071720F102183C001738F1871F7502110C021733172C27021733008D02
DATA E1FA 00 01A0 001726F28406070217331733062017331723072000002D2827251E1814120801
DATA 1236 00 01C0 E32F163717331723F204160F01173F17200F01173817 20F102183C001738F187
DATA 7244 00 01E0 1F0C02174717443D7F1741F2814C350117373502002D2622201912100C0A0301
DATA E829 00 0200 E32F1667173D0C00174017 3B0F0017401735F202043C0017400C00166717350C
DATA D638 00 0220 01166616671C00165F008D0000F201076D0000000024201E1A18140D08070501
DATA 8121 00 0240 E3301698F2818DE2020106201747172 33F011740F1021DF28771350117373502
DATA 5719 00 0260 173D0C00174017380F0017401735F202043C00174000302927232 11D190F0809
DATA 1677 00 0280 E33116CA1C0016A8002C001730003A4017303D001730F2012E340217330C0017
DATA 9CCF 00 02A0 3017353F011730F28239D20101E202012C0016CF003A002F221E1C18110D0803
DATA 8D93 00 02C0 E33116FC4016CF7D0000F1811C350217333501173 7E202010620174717233F01
DATA 5CA7 00 02E0 1740F1024EF287000702174717 47F28709F287060C02000029271D1917100C02
DATA 50A4 00 0300 E332172F1747172F3601171E7502068C020017470E01171E1722C2010000C202
DATA B6DD 00 0320 0000C0870000000100 12F1F0F0F0F0F0F1F2F5F6F0F0D100000015130F070301
DATA 92D6 00 0340 C514E62F00000000000000000000000000000000000000000000000000000000
DATA 3CAA 00 0360 0000000000000000000000000000000000000000000000000000000000000000
DATA C005 00 0380 615C000000000000000000000000000000000000000000000000000000000000
DATA C1D7 00 03A0 0000000000000000000000000000000000000000000000000000000000000000
DATA EFAB 00 03C0 4040404040404040404040404040404040404040404040404040404040404040
DATA 307C 00 03E0 4040404040404040404040404040404040404040404040404040404040404040
END  E1D3
```

Generating Random Numbers

by Teresa Elms

Code on diskette:
RPG subroutine RANDOM

Many business applications require that a programmer have access to a computer-generated sequence of pseudorandom numbers. For example, decision support systems that use mathematical models include probabilistic elements that can be simulated by pseudorandum number sequences to draw representative samples. And applications programmers use pseudorandom number sequences as test data to exercise the modules of a new application.

Unfortunately, RPG includes no built-in random number generator to produce pseudorandom number sequences. RPG differs in this regard from other high-level languages such as FORTRAN and BASIC, which, on most systems, include a predefined random number function in a subroutine library. Forced to code their own routines, RPG programmers turn to *ad hoc* methods with little theoretical support — for example, dividing the system date and time by a large prime number to generate an irrational fraction that is then treated as a random number. Or they adopt algorithms like Von Neumann's center-squares method, in which a number is squared and the center digits are extracted as the random value. But the number sequences produced by these methods repeat themselves quickly or contain undesirable number patterns.

A more effective random number algorithm is the *subtractive method* described by Donald Knuth in his book *Seminumerical Algorithms* (page

171). Knuth's subtractive method generates a large quantity of unique numbers before repeating itself. Furthermore, the generated number sequences pass common statistical tests for randomness.

The RPG subroutine RANDOM (Figure 16-47) implements Knuth's subtractive method using three modules of code. The nested subroutine RND#1 (lines 36 through 61) initializes random number array R# the first time the RANDOM subroutine is called by an application program. Nested subroutine RND#2 (lines 68 through 88) then uses the values in array R# to calculate 55 numbers of a pseudorandom sequence. RND#2 stores those numbers in array R#, replacing the previous values; these 55 numbers become the basis for calculating the next 55 numbers in the sequence when RND#2 is called again. The subroutine mainline (lines 21 through 29) determines when to execute RND#1 and RND#2 as it performs its primary function: to select one random number from array R# and return it to the calling program in the field RANDUM. The random value is expressed as a nine-digit decimal fraction between zero and one.

Let's look at each module in more detail. For the subtractive method to generate a sequence of numbers with the random properties we want, the first 55 numbers in the sequence must be chosen properly. Subroutine RND#1 performs this task by initializing a 55-element array (R#) with the sequence defined by:

$$X_{n+1} = X_{n-1} - X_n$$

where X_n represents the *n*th number in the sequence. Restated in English, each number in the sequence is obtained by taking the difference of the preceding two numbers. This initial number sequence shares some of the properties of the well-known Fibonacci sequence (i.e., the sequence 1,1,2,3,5,8,13,21,34,55,...), in which each new number is the sum of the preceding two numbers. The first two values in the sequence — the "seed" values on which the first subtraction is performed — are the integer 1 and the first nine digits of the system time and date (line 39). If the difference calculated by RND#1 is negative, the routine adds 10^9 to the nine-digit result, which converts the negative number to a positive number expressed in ten's complement forms (line 50).

Notice that these initial values are not loaded sequentially into the array; therefore, the array indexes do not correspond to any element's ordinal position in the number sequence. RND#1 multiplies the loop counter (R2#) by 21 and then divides it by 55 and obtains the remainder to calculate the next array index to use (lines 45 through 47). Multiplying the array index by 21 scatters the initial values throughout the array. The division/remainder calculation ensures that the resulting array index falls in the numerical range of one to 54 — the allowable range for a 55-element array. And because 21 is relatively prime to 55, the calculated index is never zero.

Once RND#1 initializes array R# with a Fibonacci-like number sequence, it makes three calls to the calculation subroutine RND#2, which contains the guts of the algorithm. RND#2 treats the values in array R# as 55 values in a pseudorandom number sequence. From those values, RND#2 calculates the next 55 values in the sequence. The new values overwrite the previous values in the array to become the basis for subsequent calculations when RND#2 is called again. Three passes through RND#2 "warm up" the generator; that is, any initial nonrandomness is removed before the first value is returned by the RANDOM routine to the application program.

Using Knuth's subtractive method, RND#2 generates the sequence defined by:

$$X_n = (X_{n-55} - X_{n-24}) \bmod m$$

where n is greater than 55, and the modulus m equals 10^9. (In modulo division, the dividend is divided by the modulus m and the remainder, not the quotient, is the result. In RPG, the "Move Remainder" or MVR operation extracts this figure.) In other words, the equation computes the nth number in the sequence by subtracting the 24th number preceding it from the 55th number preceding it in the sequence. Because the previous 55 numbers must be known to calculate the current number in the sequence, RANDOM computes random numbers 55 at a time. The constants 24 and 55 are not chosen arbitrarily; they are special values that guarantee many unique numbers will be generated before the sequence repeats itself.

To follow the implementation of this equation in RND#2, remember that the array index values are not equivalent to the corresponding array element's ordinal position in the random number sequence. The indexes do not even reflect the *relative* ordinal position of the array elements at all times because RND#2 overwrites the elements individually. Thus, when RND#2 begins executing for the first time, array indexes one through 55 represent the first through 55th numbers in the random number sequence; but RND#2 overwrites the first element with the 56th number calculated, then the second element with the 57th number, and so on. Halfway through execution of RND#2, the 25th array element contains the 80th random number calculated, but the 26th array element still holds the 26th random number.

The loops in RND#2 use this fact to select array elements for subtraction. Notice that the expression:

$$X_{n-55} - X_{n-24}$$

computes the difference between number pairs offset in the random number sequence by 31 positions. Similarly, the loop coded in lines 69 through 77 fills the first 24 elements of array R# with the differences between number pairs that are separated in the array by 31 positions.

The loop coded in lines 79 through 87 then inverts the order of the subtraction and computes the difference between number pairs separated in the array by 24 positions. The switch is not as crazy as it looks because the subtrahend uses values computed in the previous loop. Consequently, the subtractions in the second loop also compute the difference between numbers separated in the random sequence by 31 positions. For entries in array positions 49 through 55, the second loop uses values calculated earlier in the loop, which, again, are separated by 31 positions in the pseudorandom number sequence.

Performing control functions for the RND#1 and RND#2 routines is the RANDOM subroutine mainline. The mainline calls RND#1 the first time an application program executes the RANDOM routine. The first pass through the routine is identified by a zero value in the execution count field R1#. The execution count field identifies the random number from array R# to be returned to the calling application program by the RANDOM mainline. Each call to RANDOM increments the counter. When 55 values have been used by the calling program, the mainline executes RND#2 to generate the next 55 numbers in the sequence.

Implementation

The subroutine RANDOM can be used in any RPG program on the S/36 if five conditions are met. First, the calling program must define array R# in the extension specifications with 55 elements of 10 bytes (and zero decimal positions) each. Second, the calling program must not change the values in array R# or the value of the execution counter R1#. Third, the calling program should save the values of indicators 95 and 96, if used, because these values are changed by RANDOM. Fourth, the calling program should save the values of any fields whose names are duplicated within the routine. Fifth, before including RANDOM in a program, check for duplicate tag and subroutine names. (Appropriate naming conventions and indicator usage conventions can prevent conflicts between application programs and utility subroutines.)

RANDOM can be modified to generate pseudorandom numbers larger than nine digits. The length of the array elements in R# and the lengths of fields RND03#, RND04#, RND05#, and RND08# must accommodate the number of digits in the generated random number. If n digits are used, lines 50, 73, and 83 must substitute a field name for the constant 1,000,000,000 and the initialization routine RND#1 should set the value of that field to 10^n. In line 28, the constant .000000001 must be replaced with another field name, and subroutine RND#1 should set the value of that field to 10^{-n}.

According to Knuth, the subtractive method of random number generation produces better results than most popular generators embedded in languages like BASIC and FORTRAN. The subtractive algorithm described here can be implemented in any high-level language on almost any machine because it uses only integer arithmetic between -10^9 and $+10^9$. The heart of the algorithm relies on addition and subtraction rather than much slower multiplication and division operations, making it quite fast as well.

But no random number algorithm is perfect. Critical applications should produce similar output using at least two sources of random numbers before you accept the results.

Figure 16-47

RPG subroutines RANDOM, RND#1, and RND#2

```
*. . . . 1 ... ... 2 ... ... 3 ... . . 4 ... ... 5 .. ... 6 ..  .. 7 ... ... 8
C*******************************************************************
C*
C* RANDOM NUMBER GENERATOR. Subroutine returns nine-digit random
C* number expressed as decimal fraction between zero and one
C* in field RNDNUM.  Based on subtractive method of Knuth,
C* 'Seminumerical Algorithms,' 1981, pp. 170-173.  Array R# must be
C* defined in extension specifications with 55 elements of ten
C* bytes and zero decimal positions each
C*
C* AUTHOR       - Teresa Elms
C* DATE WRITTEN - 1/28/85
C* DATE REVISED -
C*
C*******************************************************************
C*
C* Mainline. Call initialization routine on first execution of
C* routine.  Return next random number from array R# to calling
C* program.  Call calculation routine when all 55 random numbers
C* have been used.
C*
C           RANDOM    BEGSR
C           R1#       COMP 0                              951ST CALL
C   95                EXSR RND#1                            INIT ROUTINE
C                     ADD  1         R1#     20             INCREMENT COUNT
C           R1#       COMP 55                         95    ALL VALUES USED
C   95                EXSR RND#2                            CALC ROUTINE
C   95                Z-ADD1         R1#                    RESET COUNT
C           R#,R1#    MULT .000000001RNDNUM  99             CONVERT TO DEC
C                     ENDSR
C*
C*-----------------------------------------------------------------
C*
C* Initialize random number array with Fibonacci-like sequence.
C* Send is system date and time
C*
C           RND#1     BEGSR
C                     Z-ADD55        R1#                    FORCE CALC RTN
C                     TIME           RND02# 120             GET TIME & DATE
C                     MOVELRND02#    RND03#  90             9-DIGIT SEED
C                     Z-ADDRND03#    R#,55                  LOAD ARRAY ELEM
C                     Z-ADDRND03#    RND04# 100             SAVE PRIOR VAL
C                     Z-ADD1         RND05# 100             INIT CURRENT VAL
C                     Z-ADD1         R2#     20             INIT LOOP COUNT
C           RNDLP1    TAG
C           R2#       MULT 21        RND06#  40             SPREAD VALUES
C           RND06#    DIV  55        RND07#  20              THRU ARRAY
C                     MVR            R3#     20             REMAINDER 1-54
C                     Z-ADDRND05#    R#,R3#                 LOAD ARRAY ELEM
C           RND04#    SUB  RND05#    RND05#       96        CALC NEW VALUE
C   96                ADD  1000000000RND05#                 MAKE VALUE POS
C                     Z-ADDR#,R3#    RND04#                 SAVE PRIOR VAL
C                     ADD  1         R2#                    INCR LOOP COUNT
C           R2#       COMP 54                       9696FILL 54 ELEM'S
C   96                GOTO RNDLP1
C*
C* Warm up the generator.
C*
C                     EXSR RND#2
C                     EXSR RND#2
C                     EXSR RND#2
C                     ENDSR
C*
C*-----------------------------------------------------------------
C*
C* Resets random number array with next 55 numbers of a pseudo-
C* random sequence.
C*
```

```
C           RND#2     8EGSR
C                     Z-ADD1         R2#              LOW ARRAY INDEX
C           RNDLP2    TAG
C           R2#       ADD  31        R3#              HIGH ARR INDEX
C           R#R2#     SU8  R,R3#     RND08# 100    96 CALC NEW VALUE
C   96                ADD  1000000000RND08#           MAKE VALUE POS
C                     Z-ADDRND08#    R#.R2#           LOAD ARRAY ELEM
C                     ADD  1         R2#              INCR ARY INDEX
C                     COMP 24                      96961ST 24 ELEMENTS
C   96                GOTO RNDLP2
C*
C                     Z-ADD25        R2#              HIGH ARRAY INDEX
C           RNDLP3    TAG
C           R2#       SU8  24        R3#      20      LOW ARRAY INDEX
C           R#,R2#    SU8  R#,R3#    RND08#        96 CALC NEW VALUE
C   96                ADD  1000000000RND08#           MAKE VALUE POS
C                     Z-ADDRND08#    R#,R2#           LOAD ARRAY ELEM
C                     ADD  1         R2#              INCR ARY INDEX
C           R2#       COMP 55                      9696LAST 31 ELEM'S
C   96                GOTO RNDLP3
C                     ENDSR
C*
C*************************************************************
```

Sorting Packed Dates in Files

by George Applegate

Code on diskette:
Procedure PACKDATE

If you produce reports showing monthly transactions from a file with packed dates, you are familiar with the problems that the packed date field causes. The fact that the date field contains two digits per byte with a sign on the end precludes using the normal #GSORT selection criteria when selecting records by month, year, or day rather than by the full date.

The S/36 procedure PACKDATE (Figure 16-48) uses the LDA and parameters 63 and 64 to solve the packed-date problem. Procedure PACKDATE stores the comparison date in the LDA (in positions 101 through 106). Procedure PACKDATE then inspects the second digit of the month value (position 102) and substitutes a comparison value for the #GSORT selection criteria. If the input month is 08, for example, the value 79 goes into LDA positions 111 and 112. The following sort specifications (which assume the transaction date to be in positions 296 through 299 of each input record) test each record to see whether the first digit of the transaction month equals the desired input value (0 or 1). The next two IF statements write the record if the transaction month's second digit is greater than the value from LDA position 111 (7 in the example) and less than the value from LDA position 112 (9 in the example).

Procedure PACKDATE continues to loop, repeating the logic for the day and year values, ending up with the comparison values for the day in LDA positions 113 and 114 and for the year in LDA positions 115 and 116.

Keep in mind this routine depends on the arrangement of the packed date in the input file. If the date on an input record were, say, 082286, the packed field would contain 00/82/28/6F in bytes 296 through 299. The statement

```
I D 296 296
```

in the sort picks up the digit portion (the rightmost half) of the first byte, which is the first digit of the month. The statement

```
IAP 297 297
```

instructs the sort to inspect the first half of the second byte of the packed field, which is the second digit of the month value. Using the comparison values previously stored in the LDA, the sort performs the desired record selection.

Although this procedure demonstrates how to inelude the day value, it initially was designed to select reeords on month and year without regard for the day. If you want to omit the day comparison, PACKDATE can be made more efficient by omitting the lines marked with the word "day."

Figure 16-48
Procedure PACKDATE

```
***************************************************************************
***  This procedure allows the user to sort packed dates and include selected
***  month, day and year  Developed primarily for including selected month
***  and year for month end reports
***************************************************************************
// LOCAL OFFSET-1.BLANK-*ALL
// * 'Enter desired month end date when "Enter Missing Parameter" appears'
// DATE ?1R?
// LOCAL OFFSET-101,OATA-'?DATE?
***************************************************************************
***  Set the LDA positions to fill through use of parameters 63 & 64
***  First time through, position for month
***************************************************************************
// EVALUATE P63,3-102  P64,3-111
// TAG LDABEG
// IF ?L'?63?,1'?/0          LOCAL OFFSET-?64?,DATA-' 1'
// IF ?L'?63?,1'?/1          LOCAL OFFSET-?64?,DATA-'02'
// IF ?L'?63?,1'?/2          LOCAL OFFSET-?64?,DATA-'13'
// IF ?L'?63?,1'?/3          LOCAL OFFSET-?64?,DATA-'24'
// IF ?L'?63?,1'?/4          LOCAL OFFSET-?64?,OATA-'35'
// IF ?L'?63?,1'?/5          LOCAL OFFSET-?64?,DATA-'46'
// IF ?L'?63?,1'?/6          LOCAL OFFSET-?64?,DATA-'57'
// IF ?L'?63?,1'?/7          LOCAL OFFSET-?64?,DATA-'68'
// IF ?L'?63?,1'?/8          LOCAL OFFSET-?64?,DATA-'79'
// IF ?L'?63?,1'?/9          LOCAL OFFSET-?64?,DATA-'8 '
***************************************************************************
***  If all three (month/day/year are done, get out, otherwise add 2 to each
***  parm (63 & 64) and go through LDA step again
***************************************************************************
// IF ?63?/106     GOTO LDAEND
// EVALUATE P63,3-?63?+2  P64,3-?64?+2
// GDTO LDABEG
// TAG LDAEND
// IF DATAF1-SORTFILE        DELETE SORTFILE,F1
// LOAD #GSDRT
// FILE NAME-INPUT,LABEL-MAINFILE,DISP-SHR
// FILE NAME-OUTPUT,LABEL-SORTFILE,RECORDS-?F'A,MAINFILE'? RETAIN-J
// RUN
   HSORTR    15A        3X 300 N N
//   I D 296 296EOC?L'101,1'?                   Month - 1st digit
// IFF ?L'111,1'?/   IAP 297 297GTC?L'111,1'?   Month - 2nd digit
// IFF ?L'112,1'?/   IAP 297 297LTC?L'112,1'?   Month - 2nd digit
//   IAD 297 297EOC?L'103,1'?                   Day   - 1st digit
// IFF ?L'113,1'?/   IAP 298 298GTC?L'113,1'?   Day   - 2nd digit
// IFF ?L'114,1'?/   IAP 29B 298LTC?L'114,1'?   Day   - 2nd digit
//   IAD 298 298EOC?L'105,1'?                   Year  - 1st digit
// IFF ?L'115,1'?/   IAP 299 299GTC?L'115,1'?   Year  - 2nd digit
```

```
// IFF ?L'116,1'?/   IAP 299 299LTC?L'116,1'?    Year  - 2nd digit
     FND 298 298                  Year  - 1st digit
     FNZ 299 299                  Year  - 2nd digit
     FNC 296 298                  Month/day
     FNC   7  11                  Control field
     FDC   1 256                  Data fields
     FDC 257 300                  Data fields
// END
// LOCAL OFFSET-1,BLANK-*ALL
**************************************************************************
***  If you desire all records for a select month/year, regardless of day,
***  simply omit lines 43-45 (day specifications, or place a "*" after the
***  "I" so the day specifications are "I*D" and "I*P" and "I*P" (43-45)
**************************************************************************
```

Processing DUP Keys in RPG

by John Bowers

Code on diskette:
RPG code DUPCHR

A previously published Technical Corner explained the technique for allowing workstation operators to use the DUP key for duplicating an entire input field in an interactive RPG II program. While duplicating an entire field is useful, I have run across several applications that require character-by-character duplication. Let's say that a S/36 RPG data entry program for the accounting department requires a 12-character account number, but the accounting clerks would like to key in only those positions of the account number that have changed since the previous transaction. This user request can be satisfied by enabling the DUP key in the program's screen format and by incorporating the C-specs in Figure 16-49 into your RPG program.

Suppose you want to allow the DUP key to be used when inputting values into a field called SCRIN. The DUP key is enabled by coding a Y in column 34 in the screen format source member line that defines field SCRIN. To process the DUP key, the C-specs in Figure 16-49 use two arrays, SAV and INP. When the data entry program is first called, the initial value of field SCRIN is saved in array SAV.

To recognize the DUP key character, the C-specs define a field containing a hex FC (the DUP key character for numeric fields). The C-specs use the BITOF and BITON operations to put a hex FC into field HEXFC. First, all the bits in field HEXFC are set off; then, the first six bits in that field are set on. (Field HEXFC will be used in a comparison later in the C-specs.) Next, array INP is filled with the characters from field SCRIN, and each element of array INP is checked for the DUP key character. If a DUP key character is found in any position, the appropriate array element is substituted from array SAV. If a DUP key character is not found (i.e., a new value was entered), array SAV is updated to reflect the change. When all the elements of array INP have been processed, the value in array INP is moved back into field SCRIN for use by the data entry program.

If you were to implement this logic in a data entry program, users would need to key in only those digits in an account number that have changed since the last transaction. The DUP key capability would reduce the chances of error when there are multiple transactions under the same account number or when there are standard portions of the account numbers.

If you want to allow DUP key capability in other fields, you could incorporate these C-specs into a subroutine that would be called as needed. Just be sure that the field length (FLDLNG) is as long as the longest field and that you reset array SAV accordingly. To make the subroutine more general, you may want to make field SCRIN an alphanumeric field as well. In that case, instead of using the BITOF and BITON operations to build a hex FC, you would use them to build a hex 1C, the DUP key character for alphanumeric characters. (The bit configuration for hex 1C is 00011100.)

Figure 16-49

Code for DUP key processing. (This code appears in member DUPCHR on diskette.)

```
*. .     1         2 .     . 3   .  . 4   .  . 5      . 6 .     . 7  .  . 8
     C                     BITOF'01234567'HEXFC   1
     C                     BITON'012345'  HEXFC


     C                     MOVEASCRIN     INP
     C                     Z-ADD1         Y       20
     C           Y         DOWLE          FLDLNG
     C           INP,Y     IFEQ HEXFC
     C                     MOVE SAV,Y     INP,Y
     C                     ELSE
     C                     MOVE INP,Y     SAV,Y
     C                     END
     C                     ADD  1         Y
     C                     END
     C                     MOVEAINP       SCRIN
```

Redisplaying User Procedure Parameters Using the DUP Key

by Gary T. Kratzer

Code on diskette:
Procedure DUPTST
RPG program DUPKEY
Screen format member DUPTSTFM
Assembler subroutine SUBRDU

Utility DUPKEY provides a solution to a small, but nagging, problem that has mystified S/36 programmers: how to get the Dup key to redisplay parameters for non-IBM procedures. Here's the situation. A user keys a procedure name and presses the Help key (or enters the word HELP and the procedure name — e.g., HELP BLDLIBR). On the Help screen, the user then enters the procedure parameters. After the procedure is executed, the user can press the Dup key to redisplay the procedure name and parameters — if they were entered on an IBM Help screen. Unfortunately, if the procedure was a non-IBM procedure, pressing the Dup key displays only the procedure name, not the parameters that the programmer might have prompted for via the // PROMPT statement.

IBM procedures invoked via HELP are controlled by the Help Processor (i.e., IBM program $HELP in #LIBRARY), which is responsible for updating each workstation's Dup key save area, a disk sector maintained in the Workstation Work Area of the Task Work Area. Utility DUPKEY performs this same update but lets the user specify when and with what data to update the Dup key save area (up to 120 bytes in length).

The DUPKEY utility is actually an assembler subroutine with a tiny RPG program as its driver. I generally shy away from standalone assembler programs for utility functions because a subroutine with a driver program is more flexible in fitting a particular situation. You can either use utility DUPKEY's driver or write your own to suit your needs.

Program DUPKEY's only function is to read the Dup key text from the SYSIN buffer via IBM's SUBR01 and then to call assembler subroutine SUBRDU to write the text to the Workstation Dup key save area.

DUPKEY Demo

When you run procedure DUPTST, a prompt screen (Figure 16-50; see Figure 16-51 for screen format member DUPTSTFM) is displayed that requests four parameters of varying lengths. Key some data into each parameter, and then press Enter or Command key 4 to send the job to the job queue. The procedure (Figure 16-52) saves the system return code (?CD?) in parameter 64 for later testing to determine whether Command key 4 was pressed. (The return code must be saved because program DUPKEY, which procedure DUPTST will load, resets the return code to zeros.) Next, procedure DUPTST loads program DUPKEY with the procedure name and associated parameters coded as the first SYSIN line following the // RUN statement.

Figure 16-50 *DUPTST prompt screen*

```
DUPTST test prompt screen

Parameter 1  11111

Parameter 2  2222222222

Parameter 3  333333333333333

Parameter 4  44444444444444444444

CMD4-Put on job queue                             Press enter to con-
tinue
```

After running program DUPKEY (Figure 16-53), procedure DUPTST tests parameter 64 to see whether you requested job queuing. If so, procedure DUPTST is placed on the job queue, and a PAUSE message is displayed at the system console that indicates procedure DUPTST is completed. When the procedure ends, you press the Dup key at the command line and then press Enter. The procedure name and the parameters you entered are now displayed (Figure 16-54).

Customizing DUPKEY

As I said, you do not need to use program DUPKEY; you could write your own program that invokes subroutine SUBRDU to perform the update in any manner you prefer, with any data you prefer. Note, however, that when you call subroutine SUBRDU, you must supply two RLABL statements after the EXIT operation. The first RLABL statement must be a 120-byte field (no data structures allowed) that contains the data for updating the Dup key save area. This 120-byte limit exists because that's the number of bytes the command line for a workstation occupies.

The second RLABL statement is a return code that indicates whether the update is successful. There are only two possible return codes: 0 if the update is successful and 1 if the update is not successful because the job is not running at a workstation (subroutine SUBRDU does not attempt to locate the Workstation Work Area for jobs running in batch mode).

That's all there is to it. Now, with a simple addition, your procedures can behave like IBM procedures.

Figure 16-51

Screen format member DUPTSTFM

```
*..   .. 1 ..  .  2  .  .  3 ....  4  .    .. 5      . 6     .  7     . 8
      SSCREEN01           YY         Y                           DG
      D           25 329Y                                  CDUPTST test prompt screX
      Den
      D           11 7 2Y                                  CParameter 1
      DPARM1       5 71501 Y             Y          Y
      D           11 9 2Y                                  CParameter 2
      DPARM2      10 91502 Y             Y          Y
      D           1111 2Y                                  CParameter 3
      DPARM3      15111503 Y             Y          Y
      D           1113 2Y                                  CParameter 4
      DPARM4      20131504 Y             Y          Y
      D           2124 2Y                                  CCMD4-Put on job queue
      D           232457Y                                  CPress enter to continue
```

Figure 16-52

Procedure DUPTST

```
// IF JOBQ-YES GOTO JOBQ
*
// PROMPT MEMBER-DUPTSTFM,FORMAT-SCREEN01,START-1,LENGTH-'5,10,15,20'
// EVALUATE P64,4-?CD?
*
// LOAD DUPKEY
// RUN
DUPTST ?1?,?2?,?3?,?4?
// END
// IF ?64?-2004 JOBQ ,DUPTST,?1?,?2?,?3?,?4?
// RETURN
*
// TAG JOBQ
// PAUSE 'DUPTST has successfully completed'
```

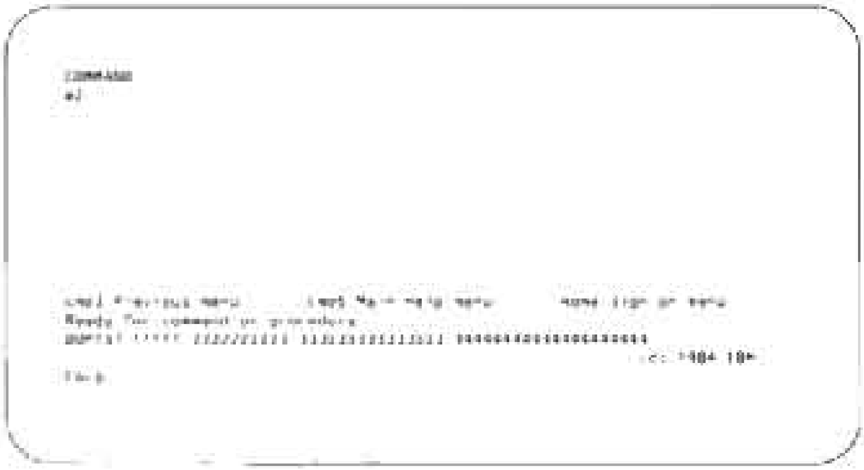
Figure 16-53

Program DUPKEY

```
    1         2         3         4         5         6         7         8
H          64                      B                                   DUPKEY
FINPUT   IP  F 120 120            SPECIAL       SUBRDU
IINPUT
I                                         1 120 DUP
C                    READ INPUT                   50
C                    EXIT SUBRDU
C                    RLABL          DUP
C                    RLABL          RCODE
C                    READ INPUT                   50
C                    SETON                    LR
```

Figure 16-54

Screen with displayed parameters



Re-creating Subroutine SUBRDU

If you don't have assembler subroutine SUBRDU, you can re-create it with procedure MKSUBRDU (you don't need IBM's Assembler Language Program Product to install SUBRDU). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRDU. You need to run MKSUBRDU only once because SUBRDU is subsequently linked into program TESTUL when it is compiled.

```
// * 'Re-creating R-module SUBRDU in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-301,DATA-'00000077'     Number of $MAINT records
// LOCAL OFFSET-309,DATA-'
 [illegible]
// LOCAL OFFSET-373,DATA-'
 [illegible]
// LOAD MAKMEM
// FILE NAME-$BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-J
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
```

Continued

```
* Patch the new SUBRDU member to insert object code
// LOAD $FEFIX
// RUN
HDR  3850 SUBRD00000
PTF  1841 RSUBRDU,99,,#RPGLIB
DATA 53AB 00 0000 E20BE2E4C2D9C4E400000001E00000000000000000000000000000000000000
DATA DD6F 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA E2CF 00 0040 E3340034F2870FE2E4C2D9C4E440F14BF04040404040340100863402008A3408
DATA F0D8 00 0060 008EF4000A3807009735A100977C40FF5CFEFEFF3502008EB6003328241D1915
DATA 2B90 00 0080 E336006A02026C7777003CF10094F4000FBA260075A11736A10091F2811E4D02
DATA 88FF 00 00A0 040091F281164D02520091F2810E35A20097F400610722013CF0002D261E1609
DATA 8802 00 00C0 E32800930094C2A100003501008E7501054C00000940E01008E0093C2A10000
DATA 866C 00 00E0 C2A20000C087000000000000060000000000000000000000000001715110901
DATA 1C34 00 0100 E302009780009F000000000000000000000000000000000000000000000000000
DATA A404 00 0120 0000000000000000000000000000000000000000000000000000000000000002
DATA 2A8A 00 0140 E33A01D9E2E4C2D9C4E4406040C39697A899898788A3404D835D40F1F9F8F96B
DATA 7DAC 00 0160 40C78199A840E34B40D29981A3A985994040C193934099898788A3A240998500
DATA 33D0 00 0180 E30601DFA28599A585840000000000000000000000000000000000000000000000
DATA 3298 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 1719 00 01C0 C50000DF00000000000000000000000000000000000000000000000000000000
DATA 37E2 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
END  E83A
```

Running Procedures in Parallel

by Ed Froste

Code on diskette:
Procedure WAITON

Have you ever thought you could reduce execution time by executing two or more job steps at once? You can use the EVOKE OCL statement, but it's difficult to use the output of the EVOKEd procedure in subsequent job steps. Procedure WAITON in Figure 16-55 lets you start two or more procedures at once and then "wait on" those procedures so you can subsequently use their output.

Figure 16-56 shows how you might use procedure WAITON. Let's say you have three steps (SORTM1, SORTA1, and SORTT1) that you want to execute at the same time. Place two of those steps (SORTM1 and SORTA1) in separate procedures, and EVOKE those two procedures from the main procedure. After you have EVOKEd those two procedures, execute the first step in the main procedure (SORTT1) and then call the WAITON procedure twice — once to "wait on" SORTM1 and once to "wait on" SORTA1. Procedure WAITON returns control to the main procedure when the EVOKEd procedures are done, and their aggregate output can be used in the next job step (UPDATE). As Figure 16-56 illustrates, procedure WAITON can be used more than once, so you can EVOKE several procedures and "wait on" each. I managed to cut execution time almost in half when I used the procedure in Figure 16-56 instead of a standard procedure that first did the SORTM1, then the SORTA1, and so on.

Figure 16-55

Procedure WAITON

```
* WAITON Proc    - form WAITON Proc-name
// TAG A
// IFF ACTIVE-?1? GOTO EX
// WAIT INTERVAL-000200
// GOTO A
// TAG EX
// RETURN
```

Figure 16-56

Sample use of procedure WAITON

```
// * 'Start of Job stream'
// EVOKE SORTM1
// EVOKE SORTA1
SORTT1
WAITON SORTM1
WAITON SORTA1
UPDATE
// EVOKE REPORT1
// EVOKE REPORT2
REPORT3
WAITON REPORT1
WAITON REPORT2
PRINTX
CLNUP
//   * 'End of Job stream'
```

Explanation of SUBR95

answered by Mike Patton

Q What is the on-line Inquiry subroutine SUBR95, and how do you use it? It appears to be called in a program if the user takes menu option 4 at the Inquiry screen. Is this correct?

A When an RPG program calls SUBR95, the subroutine tests to determine whether the "Inquiry latch" has been set by the user. The user sets the Inquiry latch when he or she takes option 4 — "Set inquiry condition for program" — at the Inquiry menu.

Subroutine SUBR95 is not called automatically when a user selects option 4. Instead, you must code an explicit call to SUBR95 in your program whenever you want to test for this condition. The format for calling SUBR95 is

```
EXIT SUBR95
RLABL          INxx
```

where *xx* is any indicator you wish. When SUBR95 returns control to the calling RPG program, the indicator referenced in the RLABL statement turns on if the operator has selected Inquiry option 4 since the last time the subroutine was called; otherwise, the indicator turns off.

The SUBR95 subroutine is useful when your program performs a time-consuming operation such as searching a disk file. By periodically testing for the Inquiry latch, you can give the user a way of escape; that is, you can let the user cancel the long-running operation *without canceling the entire program*, simply by taking Inquiry option 4.

You should not call SUBR95 too frequently because it consumes processing resources. For example, when searching a disk file, you might call SUBR95 after processing every 100th record to minimize the overhead due to SUBR95.

Flagging NEPs to Go to End-of-Job

answered by Jeff Silden

Q We need the ability to cancel all NEPs (Never-Ending Programs), compress the disk, and power off the machine without operator intervention. Is there a way to identify and cancel all NEPs?

A There is no system function to seek out and destroy NEPs, but you can use a technique that involves creating a "flag" that instructs the NEP to set on an LR indicator and go to end-of-job. The system operator can have a procedure that calls the NEP with this flag. The MAPICS NEP, AMZ00, uses this technique. The procedure call

```
AMZP01 ,,,,,,,,,N
```

ends MAPIC's NEP within about one-half second, without operator involvement.

Figure 16-57 shows the RPG code AMZ00 uses to gain operator control of the MAPICS NEP. The calling procedure passes the input parameters to the program as "workstation" input data by setting the procedure attribute for "program data in include statement." (See the *SSP Reference Manual* (SC21-9020), pages 2 through 10, for more information on this attribute.) Lines 75 through 84 of Figure 16-57 illustrate the coding for the input data. The C-spec on line 190 sets on indicator LR if the field PDMOD (the 10th parameter) is equal to N. The next C-spec line branches to the end of the detail calculations if indicator LR is on.

Figure 16-57

Partial MAPICS program AMZ00

```
*. .     1 . .   2 .   . 3 . . .  4 .  . 5 .     6 . .  . 7         8
0058 FWORKSTN CP  F      87             WORKSTN
0059 F                                               KNUM       02
0060 F                                               KSAVDS DSAVE
0061 F                                               KID    WRKID
00611F                                               KIND       99
0073 IWORKSTN NS  11    1 CZ   2 CY   3 CX
0074 I                                         4   7 PWORD
0075 I        NS  12    2 C,
0076 I                                         1   1 PAPCD
0077 I                                         3   6 PMENU
0078 I                                         8  12 PLOGS
0079 I                                        14  150PFNUM
0080 I                                        17  17 PDMOD
00842I                                        33  33 PDECD
0085 I        NS  29
0188 C* TEST FOR SHUTDOWN                                                 *
0189 C*                                                                   *
0189 C           PDMOD     COMP 'S'                      48
0189AC           PDMOD     COMP 'X'                      52
0190 C           PDMOD     COMP 'N'                      LR
0191 C   LR                GOTO END
0192 C                     SHTDN                     20
```

Setting "Log OCL" Procedure Attributes

by Dan Stephens

Q I recently installed a canned software package on my 5360. The majority of the procedures in the package have the "Log OCL statements" attribute set to Y. Is there any way — short of editing every procedure — to change this attribute for all the procedures in the library?

A From the POP library names screen (type LIBR, and press Enter), place the operation code H next to the library name. You are then prompted to select logging to be on or off. You can also do the same for individual procedures from the library member names display.

SSP Procedure Messages

by Alvaro de Leon

The following is typical of the messages displayed when a series of SSP procedures are run in sequence:

- COPYDATA procedure executing
- DELETE procedure executing
- RENAME procedure executing

But sometimes you need to be able to see at a glance the file name or other parameters the executing procedure is using. If you alter one line of your SSP procedures, you can add more informative messages to them. Most SSP procedures contain the following lines:

```
// IF EVOKED-NO
   IF JOBQ-NO*nnnn
```

where *nnnn* is the message identification code for the message corresponding to the procedure. If you change the *nnnn* portion of this statement to

```
procname?1?,?2?,...,?n?
```

where *procname* is the name of the procedure and the substitutional expressions are the procedure parameters, you can display the following messages in an example session:

```
* COPYDATA  PAYMAST,,PAYMASTN,...,REORG,OMIT,1,EQ,'*',....
* DELETE    PAYMAST,F1,.....
* RENAME    PAYMASTN,PAYMAST,
```

The new messages also appear on the history file — an advantage when you are tracing problems.

Note that to use this technique, you must modify every SSP procedure from which you need more complete information. Because a new release of the SSP will replace your modified procedures, be sure to maintain documentation so you'll know which procedures to modify in the new release.

Displaying Error Messages Without Message Members

by Larry N. Forrister

Code on diskette:
Procedure ERROR

I have found a way to display messages without placing the message text in a message load member. This lets me conveniently issue impromptu messages with operator options from my procedures. I call procedure ERROR (Figure 16-58), passing these two parameters:

```
ERROR 03,'Select 0 to continue;  3 to cancel job.'
```

Parameter 1 specifies any of the standard options (i.e., 0, 1, 2, or 3); you must specify at least one option. Parameter 2 is the message text, which must be enclosed in single quotation marks if it contains embedded blanks. Only the first 72 out of a maximum 75 message characters are displayed.

Following a call to procedure ERROR, the operator's response to the message can be tested with the ?CD? substitution expression in an IF statement. Responses 0, 1, and 2 correspond to the ?CD? values 1010, 1011, and 1012 respectively. Response 3 causes an immediate job cancel.

Figure 16-58

Procedure ERROR

```
// LIBRARY NAME-O
// LOAD $CPPE
// RUN
// ERR ALPHA-SYS,MIC-1366,CONTROL-?1?,VARIN-'?2?'
// END
```

Using SSP's ERR Procedure to Display User Messages

answered by Bob Tipton

Q User confusion about the right response to take to a message abounds in our S/36 shop. This situation is due in part to our use of the // PAUSE statement. We use the PAUSE statement to alert users to situations such as conflicting jobs running or the need to insert a diskette in a particular slot. The PAUSE statement forces a user to respond to the message with a zero (0), and then, depending on the reason the message was sent to the user, the procedure continues or is canceled, as appropriate.

Unfortunately, users have become accustomed to taking the zero option to // PAUSE statements and consequently take the zero option to other, unexpected messages like DUPLICATE KEY FOUND. Is there a way I can cause a message to be sent to the user with options other than zero (like option 3) for messages that indicate a terminal situation (e.g., a conflicting job running)?

A A perfect solution to your problem exists, and this solution will present a consistent format for all messages (your messages and system messages) sent to users. The IBM-supplied S/36 ERR procedure, in combination with a user-defined, level-one message member, gives you the capability of sending messages to users with options 0, 1, 2, or 3.

To use procedure ERR, you first must create a message member. This level one message member can be called anything meaningful to you — something like USERMSG for all user messages or APMSG for accounts payable specific messages. Key the message member following the example in Figure 16-59a.

After you have keyed in the message member, use the CREATE procedure to create a message member load member. For the message member in Figure 16-59a, you would use the following CREATE statement:

```
CREATE USERMSG,REPLACE,library name
```

To use the message member in Figure 16-59a to display messages to your users, you must key the statements in Figure 16-59b into your procedures. Then, the message in Figure 16-59c will be displayed to your users when a conflicting job is running.

If you want to add variable data to your messages to help describe the message (e.g., if you want to add the name of the conflicting job to the message in Figure 16-59a), follow the example in Figure 16-59d. When procedure ERR interprets the message in Figure 16-59d, the pound signs are replaced with the job name specified in the third parameter, and the message in Figure 16-59e is displayed.

Procedure ERR lets your procedures display messages and issue options in the same way the system procedures display messages and issue options. Thus, if you use procedure ERR, your users will see consistency in all messages, and they will no longer simply take the zero option because that is the way they always have done it.

Figure 16-59a

Sample message member

```
USERMSG,1
0001 There is conflicting job running
0002 Insert diskette ABC into slot 1

nnnn  .  the last message in the message member
```

Figure 16-59b

Sample // MEMBER statement to use in procedures

```
// MEMBER USER1-USERMSG,LIBRARY-library name

ERR 0001,3
```

Figure 16-59c

Sample displayed message

```
USER-0001 (  3)

There is a conflicting job running
```

Figure 16-59d

Variable data added to a message

```
USERMSG,1
0001 Job ######## is running now and conflicts with yours

// MEMBER USER1-USERMSG,LIBRARY-library name

ERR 0001,3,job name
```

Figure 16-59e

Sample displayed message with variable data

```
USER-0001 (  3)

Job job name is running now and conflicts with yours
```

Security

CHAPTER

17

Using S/36 Security

by Matthew Henry

Your S/36 probably contains most information essential to running your organization: accounting figures, payroll statistics, sales and production history, inventory records, and — if you use the office management features of SSP — most of your intra-office memos, letters, and scheduling. Losing this critical asset because of equipment failure or other calamity would be disastrous, so naturally you protect the information on your system by following a strict backup procedure.

However, backing up your data doesn't protect against a loss that can be just as devastating: undiscovered disclosure or alteration of sensitive corporate records. Such a loss is often insidious; you may not discover the damage until it's too late to repair — if you discover the damage at all. To help you protect your corporate data from unauthorized access, the S/36 incorporates a three-level security system: user ID, password, and resource. Each level provides a layer of protection, and each requires effort on your part to implement and manage.

It's a well-known fact that no computer security system provides absolute protection. But, by investigating the strengths and weaknesses of S/36 security options, you can choose the features that provide the level of protection you need and that guard against common pitfalls that might leave your system open to compromise. To understand how S/36 security works, you need to learn about its major components: user IDs, user profiles, passwords, security classes, and authorities. This article describes these components and shows you how they work together to provide the three levels of S/36 security.

User IDs

From a user's standpoint, S/36 security begins with the sign-on screen (Figure 17-1). To gain access to the system at this lowest level of security, a user simply enters an eight-character name, called a user ID, on the sign-on screen. Along with the user ID, the user can optionally specify a menu name, library name, or procedure to use after sign-on. The user ID, which provides SSP with a handle for keeping track of users' work on the system, can be any combination of alphameric characters, as long as the first character is a letter, digit, or $, @, or #. Embedded commas and blanks aren't permitted. The SSP accepts any user ID that follows these rules, gives the user immediate access to the system, and logs the date and time of sign-on as well as various actions the user takes during a session. Thus, the first level of S/36 security provides only an audit trail of user activity; it doesn't control which users may access the system.

Figure 17-1

Sign-on screen

```
                         SIGN ON                        W1
                                                 Optional-*

          User ID                      SMITH

          Password

          User menu                                      *

          Library                                        *

          Procedure                                      *



     Help-Assistance for sign on
                                  COPYRIGHT 1985 IBM Corporation
```

Figure 17-2

User profile screen (user ID security level)

```
                           SECEDIT USERID                      W1
                                                      Optional-*
             Edit the user profiles in the user identification file

        Mode  Browse or Update - Key in changes and press Enter

   User ID                                              SMITH

  Default user menu                                     PAYMEN    *

  Default sign-on procedure                             TIMECARD  *

  Default library                                       PAYROLL   *

  Beginning help menu                                   MAIN      *

  Comment                                        PAYROLL CLERK    *


    Roll keys-Page          Cmd2-Scan             Cmd3-Restart
    Cmd4-Remove             Cmd5-Add mode         Cmd7-End
  COPR IBM Corp 1985
```

At the level of user ID security, a user profile is optional. You use it to store user preferences, purely as a convenience for the user. To create this record for each user, you use the SECEDIT procedure to set default values for the menu name, library name, and procedure (Figure 17-2). If the user leaves these fields blank on the sign-on screen, the SSP substitutes the default values from the user profile. Because at this security level any user-entered values override the defaults, the user profile is not a security feature.

In addition to default values for the sign-on screen fields, the user profile lets you specify the beginning help menu for the user. The beginning help menu appears immediately after sign-on (unless you've already specified a default user menu) and whenever the user requests system help by pressing Command key 6. Users can change their beginning help menu in the user profile by pressing Command key 23 while displaying any help menu.

Password Security

Although user profiles are an optional convenience under user ID security, they are required for the other two levels of security, one of which is password security. With password security, you assign each user ID a corresponding four-character password. On the sign-on screen, the user keys both the user ID and password. The password (which does not appear on the screen when typed) must match the value stored in the user profile or else the user is denied access.

Figure 17-3

Activate password security screen

```
                    SECDEF  Activate password security                         W1

               Type any changes and press the Enter key to schedule
               password security to be activated at the next IPL

 Master security officer user ID                                   GEORGIA

 Master security officer password                                  WR3F

 Override user ID                                                  GEORGIA

 Override password                                                 WR3F

 Maximum invalid sign-on attempts                          3 - 25  03

 Start password date checking to
   require users to change passwords?                         Y,N  Y

         Press the Enter key to schedule password security to be activated

Cmd3-Display previous menu          Cmd7-End               COPR IBM Corp  1985
```

Before you can assign passwords, you must activate password security by running the SECDEF USERID,ACTPW procedure, which brings up the Activate password security screen (Figure 17-3). You must assign one user ID to be the Master Security Officer (MSO) for your installation. The MSO has universal access to everything in the system, so be sure to guard this user ID and password carefully. You must also choose an override user ID and password, which can be the same as the MSO user ID and password. The override user ID and password lets you sign on to the system if the security files are damaged or destroyed.

To activate password security, you also specify the maximum number of sign-on attempts permitted and whether you want to use password date checking, which I describe later. The maximum number of sign-on attempts limits the number of consecutive times a user can enter an invalid user ID or password. Each invalid attempt is logged with a message to the console operator, and after the maximum number of attempts, the workstation is varied off-line, requiring console or system operator intervention.

Password User Profiles

With password security active, the user profile looks different (Figure 17-4). You now must specify a password, security classification, service aid authority, and optional badge number. The password must be a full four characters long and can consist of any characters except embedded blanks. You may choose to assign passwords centrally or let users choose their own passwords. Because passwords are the linchpin of your security system, you should carefully choose your strategy for managing them. Later, we'll discuss the factors you need to consider when deciding on a password policy.

The security classification lets you limit the general powers of various users. There are five security classes: Master Security Officer (M), Security Officer (S), System Operator (O), Subconsole Operator (C), and Display Station Operator (D). You're free to put as many users as you like in each of the classes, although each user can be a member of only one classification. You established the MSO when you activated password security.

The S classification lets you deputize other users to manage security functions, such as creating and revoking user IDs, assigning passwords, and changing user authority — but not changing the MSO or other S-class user profiles. S-class users can also sign on at the system console and execute system operator commands, such as those controlling jobs, devices, and spool-

Figure 17-4

User profile screen (password security level)

```
                              SECEDIT USERID                                W1
                                                                    Optional *
                 Edit the user profiles in the user identification file

          Mode  Browse or Update - Key in changes and press Enter

   User ID                                                        SMITH
   Password
   Security classification                              M,S,O,C,D  M
   Service aid authority ?                                    Y,N  Y
  Badge number                                   00000000-99999999          *
  Comment                                               M I S  DEPT  HEAD   *




   Roll keys-Page             Cmd2-Scan               Cmd3-Restart
   Cmd4-Remove                Cmd5-Add mode           Cmd6-Show passwords
   Cmd7-End                   Cmd9-Additional information
  COPR IBM Corp 1987
```

ing. The O classification excludes all security functions, but lets the user perform the system operator functions previously described. C-class users can access subconsoles and enter spooling and device commands associated with their assigned subconsole, but they can't sign on to the system console or use job control commands. D-class users can run their own applications. Users in any class can list their own security information (except password), change their password, or secure their own files, folders, or libraries.

Service aid authority lets a user run low-level maintenance procedures that could compromise security (i.e., the DUMP, PATCH, DFA, and PTF procedures). Because a sophisticated user could employ these tools to subvert the security system, either by gaining access to passwords or by modifying IBM modules that control security, you should take care in granting this privilege. Generally, only the MSO and System Operators need this capability so they can collect information when reporting SSP problems to IBM.

Figure 17-5
User profile screen (additional parameters)

```
                              SECEDIT USERID                             W5
                                                                  Optional-*
           Edit the user profiles in the user identification file

User ID      SUBBY


Default user menu                                                         *
  Menu mandatory?                                       Y,N    N
Default sign-on procedure                                                 *
  Procedure mandatory?                                  Y,N    N
Default library                                                           *
Beginning help menu                                            MAIN       *
Allow user to create folders?                           Y,N    Y
  Maximum folder size in blocks                 4-65535,NOMAX  NOMAX      *
Comment                                             M.I.S  DEPT  HEAD     *



Roll keys-Page       Cmd2-Scan          Cmd3-Restart
Cmd5-Add mode        Cmd7-End           Cmd10-Previous information
                                                      COPR IBM Corp  1987
```

Badge security, activated at the same time as password security, requires terminals with a magnetic strip reader to scan a badge before a user can gain access to the system. Thus, badge security can provide additional security in remote locations.

Pressing Command key 9 on the user profile screen displays a screen of additional parameters (Figure 17-5). You're already familiar with the default user menu, sign-on procedure, library, and beginning help menu values. Under password security, however, you can prevent user overrides of the menu and procedure defaults by specifying mandatory menus and mandatory procedures. With mandatory menu control, a user can select only from the menu options you provide; the user can't enter *ad hoc* procedure commands and can't access the system help facility to prompt for and to run commands.

The final user-profile parameters keep users from creating document folders or limit the size of folders they create. This feature, added in SSP Release 5.1, keeps users from inadvertently using up disk space by unintentional folder creation.

Password Security Strategies

A chain always breaks at the weakest link. For S/36 security, that link is the password. Unless passwords are kept secret, you are secure only in your mind. Passwords can be compromised in one of two ways: a user indvertently or deliberately discloses the password, or an interloper guesses the password. You can reduce the chance of accidental disclosure by requiring that passwords be memorized — never written down. The policy should also prohibit users exchanging passwords or signing on with another's user ID.

To reduce the chance that someone might guess a password, you should choose password character combinations at random. The combination should include both numeric digits and special characters and should not follow a pattern (such as inverted phone number digits) or use mnemonic tricks (such as companies that require an employee's password to be his or her mother's maiden name). X3$R is much harder to guess than FRED or MARY, but it is also, unfortunately, much harder to remember; you have your work cut out for you enforcing the no-written-password policy when using such passwords.

However, the risk of using less reliable passwords is considerable. If you follow a pattern and the pattern is disclosed, all your passwords are compromised in one blow. Similarly, mnemonic passwords are among the first that an intruder will try in a guessing attack. You should also regularly check for user ID/password combinations programmers commonly use during testing, such as GUEST, TEST, USER, and the like. These are high up on the list of candidates for a guessing attack.

One strategy you might consider is letting users assign their own passwords with the PASSWORD procedure. When users run the PASSWORD procedure, the system prompts them for their current password and then prompts for the new password twice as verification. Encourage users to change their passwords at regular intervals so if a password is compromised, the duration of its value to an intruder is limited. You can still review user-selected passwords to cull poor choices. And if a user forgets a password, you can change the password to a new, known value by signing on as the MSO. Letting users choose their own passwords reduces your password management workload and gives users an opportunity to select passwords that are not obvious, but are still easy for them to remember.

Password date checking, mentioned previously, is a useful password security feature that lets you enforce mandatory password changes at regular intervals. You specify the number of days before a password expires and, optionally, the number of days in advance to warn the user of impending expiration. The user receives the warning at each sign-on and can use the

PASSWORD procedure to change passwords at any time. If the user doesn't change the password by the expiration date, the password becomes invalid, and a security officer must intervene to change it.

Protecting Resources

In addition to user ID security and password security, the S/36 provides resource security to protect files, libraries, folders, directories, and special resources by controlling access to them individually. You should use resource security when mandatory menu security isn't practical (e.g., when users need to execute SSP procedure commands). Be aware, however, that resource security exacts a significant cost — in both management effort and system performance. You activate resource security by running the SECDEF RESOURCE,CREATE and SECDEF RESOURCE,ACTRES procedures, which create the resource security file and set resource security to begin at the next IPL. The resource security file contains the security profiles for each object you want to protect.

Resource security is built on six access levels: owner, change, update, read, run, and none. For files and libraries, you assign each secured resource a default access level. Then you define each user who is to have a different access level than the default level (Figure 17-6). For folders, you establish authorization lists to define user access (Figures 17-7 and 17-8). In addition to securing files, libraries, and folders, you can secure groups of each. For example, you could set up a resource group called ADM. You then could secure a library named ADM.LIB, a file named ADM.MST, and a folder named ADM.FLD with one group resource record.

When someone attempts unauthorized access to a secured resource, an information message is sent to the system console as well as to the user. The message indicates which resource could not be accessed. When you define a resource record, you can specify that the system log every access to the system history file, whether successful or unsuccessful. Then you can periodically analyze the history file to collect resource use statistics.

Securing Files

Resource security lets you secure any type of data file and alternate indexes. Access levels for files include five of the six available levels (file security lacks the run-access level). Assigning owner access to a user ID lets that user create, rename, or delete the file and also read, write, update, or delete records in the file. Change access lets the assigned user create or delete the file as well as read, write, update, or delete records. Update access gives access only to individual records; the user cannot delete, rename, or create the file. Read access limits the user to viewing the file's records, and an access level of none prevents any kind of access to the file or its records.

You secure an alternate index differently than you do a file. To create an alternate index over a physical file, the user must have at least read access to the parent. The resource record for the alternate index must have the parent file listed as the parent resource. If you assign a user a higher access level to an alternate index than to the parent, it is possible, depending on the program, for the user to gain a higher level of access to the parent file than you want.

Securing Libraries

You can secure entire libraries, including the system library (#LIBRARY), but you cannot define a separate level of security for each type or kind of library member. If you assign the owner-access level to a library, you allow a user complete control over the library. The user may create, rename, or delete the library as well as create, change, run, list, remove, or copy any member of the library. Update access prevents the user from renaming the library but lets the user create, change, run, list, remove, or copy any member in the library. Read access lets the user view any member; run access lets the user access the library member for running only — not for viewing. Only security officers can secure system library #LIBRARY. Although you cannot prevent a user from running members in the system library, you can control editing, copying, displaying, and deleting members.

Securing Folders

Using resource security, you can secure folders for DisplayWrite/36 (documents), Personal Services/36 (mail and mail logs), and IDDU data dictionaries. In addition to entire folders, you can secure individual subdirectories and documents and PC Support/36 virtual disks. However, resource security for folders is different from file and library security; you use authorization lists for folders, subdirectories, and folder members.

An authorization list includes a group of user IDs and their assigned access level. In addition to the required primary authorization list, you are allowed an override list. Before you can access a folder, the system goes through a security check routine. First, the system checks the resource file to determine whether the folder or directory is secured. If so, the system next scans the override authorization lists for a name match. If your user ID and access level are high enough to perform the required open and subsequent action, access to that folder or directory is granted. If the system does not find an override authorization list, the system checks the resource file to see whether the folder is part of a group. If no override authorization list or group match is found, the system then searches the primary authorization list. If a match is found and your access level is high enough, you get access to the folder. If a primary authorization list is not found, the system uses the default folder access level.

Only document or text folder members can have individual security; however, you must secure the folder or directory first. After access to the folder or directory is cleared, the individual document is checked for security and access levels.

Owner access for a folder lets the user change or remove security; rename the folder; add, change, or remove security information for folder members or directories; create or remove the folder or directory; and read or change any information in any member of the folder. Change access lets the user change or remove security for folder members owned by the user, secure any folder member not already secured, create folder members, delete or create the folder or directory, remove folder members assigned to the user as update, and read or change information in any folder member.

Update access lets a user change or remove security information for members owned by the user, secure any member not already secured, create a new folder member, remove any member assigned to the user, and read and change any information in members of the folder or directory. Read access lets users read folder members and copy information from the member if their user ID has read access for the entire folder or directory. Run access applies to PC shared folders and permits a PC user to run a file contained in the secured folder or directory.

Be careful when securing individual documents in a DisplayWrite folder. For example, if you assign the person responsible for backing up folders an access of none, an error occurs when the system tries to back up that document.

Resource Security Overhead

To implement and maintain resource security may require a significant management effort. Moreover, resource security adds significant processing overhead to all your jobs. Whenever a program is initiated that uses secured resources such as files and libraries, the resource security file must be searched to see whether the user is allowed access. Each resource check requires at least two disk accesses. Whether this processing overhead will degrade your interactive response times depends on the type of programs you run. For batch jobs, the time required for resource security checks is usually insignificant because batch jobs infrequently initiate new programs. On the other hand, interactive jobs may perform frequent program initiations as the user moves from function to function. If an interactive program processes many files, the time required for resource checking may slow initiation perceptibly, resulting in degraded response times.

MRT programs that remain active between requesters are an exception because the security check is performed only once — when the first requester initiates the program. Subsequent requesters won't go through the security check because the MRT job is already active. Thus, another security consideration is controlling access to MRT programs separately from securing the MRT programs' resources. You can either control access to the library containing the MRT program or use mandatory menus to prevent direct user execution of MRT procedures. If you don't provide this extra control, an unauthorized user could become a requester on a MRT program that accesses sensitive files, compromising your security.

Figure 17-6

File and library security-access-level screen

```
                    SECEDIT RESOURCE                               W1

                  Edit the resource security file

      Mode  Add - Key in requested information and press Enter or Cmd6


                                                          Optional-*
Resource name                                             PAYMAST

Special resource type                              A,G,S          *

Default access                                 O,C,U,R,E,N  N

Parent resource name                                              *

   Is the parent resource a group resource record?      Y,N  N
Log successful accesses?                                Y,N  N

Roll keys-Page        Cmd2-Scan        Cmd3-Restart
Cmd5-Add mode         Cmd6-Display user records          Cmd7-End
                                                      COPR IBM Corp  1985
```

Figure 17-7

Folder security screen

```
SECURE A FOLDER
  Type choices, press Enter
   ITEM                                    CHOICE      POSSIBLE CHOICES
   Folder name                             PAYMEMOS
   Default access                          N           O=Owner
C=Change
                                                       U=Update     R=Read
                                                       N=None
   Primary authorization list              PAYDEPT     Name of list
   Override authorization list             MASTER      Name of list
   Log successful accesses?                2           1=Yes     2=No





 Cmd3=Go back      Cmd7=End
 Cmd9=Work with primary authorization list
 Cmd10=Work with override authorization list
                                                          COPR IBM Corp
1986
```

Don't Miss a Step

We have reviewed the security steps you can take to control all levels of access to the S/36. These measures protect against carelessness as well as intentional damage. For example, when you specify the update-access level for all master files, the files cannot be deleted without changing the resource security record. This measure can prevent unwanted deletion of important files.

Resource security is also appropriate in a development environment. Limit access to development files and libraries to programmers, and restrict user authority to change production programs. Likewise, keep programmers from accessing production modules to prevent them from inadvertently destroying production files and altering production programs.

Many people have never used any type of security on their S/36 except for user IDs. They are missing many opportunities to protect a significant investment — their software programs and data.

Figure 17-8

Authorization list screen

```
CREATE AN AUTHORIZATION LIST
  Authorization list.        PAYDEPT

  Type user ID, press Enter to position list
   ITEM                            CHOICE      POSSIBLE CHOICES
   Position list to                            Starting character(s)

                                LIST OF USER IDS                        All
Type user ID and access level (O=Owner C=Change U=Update R=Read E=Run
N=None)

USER ID   ACCESS     USER ID   ACCESS     USER ID   ACCESS    USER ID
ACCESS
  SMITH        O        GRANT        E
  JOHNSON      C        HENIG        E
  LLOYD        U        WILSON       E
  SAWYER       R        DORSETT      N
  SWANSON      R        COMPTON      N
  DONNALY      E        FRIEND       N
  BECKMAN      U
  THOMPSON     U
  FIELDS       R
  GAITHER      E

Cmd3-Go back      Cmd7-End      Roll-Page                 COPR IBM
Corp.1986
```

Preventing a User from Signing On to Multiple Workstations

by F.R. Helmus

Code on diskette:
Procedures LOGIN, ONEUID
RPG program ONEUID
Message member ONEUIDM

S/36 utility ONEUID discourages the sharing of user IDs and prevents users from signing on to more than one terminal with the same user ID. Thus, utility ONEUID helps with security and lets you more accurately determine who's doing what on the system.

To implement utility ONEUID, you must install assembler subroutine SUBRUL (*Retrieving a Library's Users*, page 272), which displays the users of a specific library. You must also modify the SECEDIT USERID procedure to specify a mandatory sign-on procedure for each user you don't want to sign on to more than one terminal. The OCL in the sign-on procedure (procedure LOGIN in Figure 17-9) contains the name of the library in which you store utility ONEUID — in my case, TOOLKIT.

Utility ONEUID consists of procedure ONEUID (Figure 17-10), RPG program ONEUID (Figure 17-11), and message member ONEUIDM (Figure 17-12). The program requires a compile-time array with all libraries on disk. When a user signs on, the program checks all users in all libraries. If it finds a user ID that matches the one just entered, and if the user ID it finds is not signed on to the same terminal or not running from the job queue, external switch U1 is set on and control returns to procedure ONEUID. The procedure then displays an error message and ends the session. However, if the user invokes inquiry mode before program ONEUID is completed, procedure ONEUID cannot end the session. In this case, the procedure displays a message at the system console alerting the operator that the user is signed on to two terminals.

Figure 17-9

Procedure LOGIN

```
// ATTR INQUIRY-NO,CANCEL-NO              * No Cancel or Inquiry
**********************************************************************
*   Procedurename        LOGIN                                       *
*                                                                    *
*   Function             Specify some action for each user who's     *
*                        signing on                                  *
*                                                                    *
*   Note                 With SECEDIT USERID, you must specify this  *
*                        procedure as a mandatory log-in procedure   *
*                        to prevent users from bypassing this        *
*                        procedure with the Attn key                 *
*                                                                    *
*   Creationdate         23/01/'89                                   *
**********************************************************************
// ONEUID,TOOLKIT ?USER?
```

Figure 17-10

Procedure ONEUID

```
*******************************************************************
*  Procedurename          ONEUID (One User-ID)                    *
*                                                                 *
*  Call format            ONEUID User-ID   (e g . ONEUID ?USER?)  *
*                                                                 *
*  Function               Prevent users from signing on to more than *
*                         one terminal                            *
*                                                                 *
*  Creationdate           23/01/'89                               *
*  Last revision date     23/01/'89                               *
*                                                                 *
*  Author   E R Helmus    Public Domain Software 1989             *
*******************************************************************
// ATTR INQUIRY-NO,CANCEL-NO              * No Cancel or Inquiry
// LIBRARY NAME-TOOLKIT
// MEMBER USER1-ONEUIDM
// LOCAL OFFSET-201,BLANK-10
// LOCAL OFFSET-201 DATA-'?1R'0001'?'     * Pass User-Id to program via LDA
// LOCAL OFFSET-209,DATA-'?WS?'           * Pass WS-Id to program via LDA
*
* Program ONEUID turns on Switch U1 if the user is already signed on
*
*
// LOAD ONEUID
// RUN
// IF SWITCH1-0 RETURN                    * If not already signed on, return
*
// ERR 0002,2 '?L 201,10'?'               * Display Error-message
*
* If the terminal is not in inquiry-mode, sign terminal off, else,
* display message at system-console
*
// IF INQUIRY-NO OFF
// ELSE ** 'Userid ?L'201,8'? is already Signed on at Terminal ?L'209,2'?'
```

Figure 17-11

Program ONEUID

```
*        1         2         3         4         5         6         7         8
     H                                                                   ONEUID
     H*****************************************************************
     H*  Name               ONEUID - IBM S/36 Utility                    *
     H*                                                                  *
     H*  Function           This program prevents users from signing on to *
     H*                     more than one terminal                       *
     H*                     It checks the users in a list of libraries   *
     H*                     specified in the compile-time array @L       *
     H*                     If the specified user is already active on   *
     H*                     another terminal, the external switch U1 will *
     H*                     be set on  and procedure ONEUID shall display *
     H*                     an Error message and signs the terminal off  *
     H*                     If the terminal is in inquiry-mode, a message *
     H*                     will be sent to the system-console           *
     H*                                                                  *
     H*  Adjust             To adjust this program to your own environment.*
     H*                     just add your library names to the compile - *
     H*                     time array at the and of the listing and     *
     H*                     put the new number of libraries in the E-Specs *
     H*                                                                  *
     H*  Note               Before compiling this program, be sure you   *
     H*                     have already installed assembler subroutine  *
     H*                     SUBRUL, NEWS 34/38, August '87  (Thanks  Matt!)*
     H*                                                                  *
     H*  Bugs               If a user signs on at more than one terminal *
     H*                     simultaneously, no sign-on will succeed!     *
     H*                                                                  *
     H*                                                                  *
     H*  Author             E R Helmus  - Public Domain Software         *
     H*                                                                  *
     H*  Creationdate       23 Januari 1989                              *
     H*  Last revision date 23 Januari 1989                              *
     H*****************************************************************
     E*  Array @L with all library names                                 *
     E*****************************************************************
```

```
E                    @L      1  20  8
I*****************************************************************
I*  Datastructure JDBDS with information of jobs (used by SUBRUL)    *
I*
I*  Pos   1  -  8    User-Id
I*        9  - 16    Job name
I*        9  - 10    Workstation-Id of Job
I*       41  - 46    Jobqueue time (000000 if not Jobqueue)
I*****************************************************************
IJDBDS       DS
I                                          1   8 USERID
I                                          9  16 JOBNAM
I                                          9  10 WS
I                                         41  460JQTIME
I*****************************************************************
I*  Local Data Area with User-Id and WS-Id to check                  *
I*****************************************************************
I          UDS
I                                        201 208 UID
I                                        209 210 WSID
C*****************************************************************
C*  Main Line                                                        *
C*****************************************************************
C                    SETOF                     U1
C*
C          UID       IFNE *BLANKS                    If ID specified
C                    EXSR $CHECK                       Check libraries
C                    END                             End
C*
C                    SETON                     LR
C*****************************************************************
C* Subroutine $CHECK, check all Libraries for the User-ID            *
C*****************************************************************
C          $CHECK    BEGSR
C*
C          1         DO   20       L       30        Do for each library
C*
C                    MOVE 'N'      NOUSER  1           Not more users
C                    MOVE @L,L     LIBNAM  8           1st libnam
C                    Z-ADD0        X       30          Seqence  - zero
C*
C                    EXSR $LIB                         Check lib
C*
C                    END                             End Do until
C                    ENDSR
C*****************************************************************
C* Subroutine $LIB, search all users for a library                   *
C*****************************************************************
C          $LIB      BEGSR
C*
C          NOUSER    DOUEQ'Y'                        Do until no users
C*
C                    EXIT SUBRUL                      Call SUBRUL
C                    RLABL         LIBNAM
C                    RLABL         X
C                    RLABL         JOBDS
C*
C          USERID    IFEQ UID                         If user active
C          WS        IFNE WSID                         If other term
C          JQTIME    IFEQ *ZERO                         If not Jobq
C*
C                    SETON                     U1
C                    MOVE 'Y'      NOUSER
C                    MOVE WS       WSID                  Save WS-ID
C                    Z-ADD20       L                     Jumpout DO
C                    END                                End If
C                    END                               End If
C*
C                    ELSE                             Else
C          JOBNAM    IFEQ *BLANKS                      If EOL
C                    MOVE 'Y'      NOUSER
C                    END                               End If
C                    END                              End If
C                    ADD  1        X                  Set sequence up
C                    END                             End Do until
```

Displaying the VTOC Graphically

by Gary T. Kratzer

program by Mel Beckman

Code on diskette:
Procedure VGRAPH
RPG program VGRAPH
Screen format member VGRAPHFM
Assembler subroutine SUBRVR

Monitor your file, library, and folder allocations with utility VGRAPH's bar graphs.

Not long ago, in a computer room far, far away, there was a S/34. Although it was very powerful, it had a mere 256 MB of disk storage. Programmers and data processing managers everywhere fought a difficult battle to conserve quickly decreasing disk space. Then, just when all hope seemed lost, a redeemer appeared: the S/36. This new system boasted more disk space than they could possibly ever need. But with no way to track disk use, they continued to misallocate (too often *over*-allocate) space for files, libraries, and folders. Too soon they were back in the boss's office, begging for more disk.

You probably have learned, as they did, that winning the "space battle" often depends not on *more* disk space, but *less wasted* disk space. A simple solution for monitoring system disk use is VGRAPH, a utility composed of RPG program VGRAPH, procedure VGRAPH, screen format member VGRAPHFM, and assembler subroutine SUBRVR. The VGRAPH utility displays file, library, and folder allocations in an easy-to-read, bar-graph format. You can access three types of information from this bar-graph screen: the percent of system space individual file, library, and folder allocations consume, the percent of allocated space each uses, and the percent of allocated space each has still available. You can see from the percent of system space used which files, libraries, and folders have the most space allocated. You then can look at all or selected items' allocated space and decide whether the allocations are appropriate. Using this information, you can increase or decrease file, library, and folder allocations to distribute disk space correctly.

The VGRAPH utility consists of program VGRAPH (Figure 18-1), screen format member VGRAPHFM (Figure 18-2), and procedure VGRAPH (Figure 18-3).

To use the VGRAPH utility, key in:

```
VGRAPH filename,altindex (Y or N),,SORT/NOSORT
```

where:

• *filename* is either the complete or partial name of the VTOC entry you want displayed

• *altindex* is Y (display alternate indexes for the first file matching *filename*) or N (do not display alternate indexes)

• *SORT/NOSORT* specifies whether you want the entries sorted by name before they are displayed .

All of these parameters are optional. If you leave the first parameter blank, the system displays all VTOC entries. (This can be changed after VGRAPH is loaded, as explained later in this article.) Specify the second parameter only if you specified the first parameter.

After you load utility VGRAPH, the system displays a screen (see Figure 18-4) with columns showing the filename, file type (I=indexed, X=alternate index, S=sequential, D=direct, L=library, F=folder), number of records used, and number of records available. The bar graph on this screen represents the percentage of total system disk space occupied by all or specified VTOC entries. Using command keys, you can modify the bar graph to show to what extent file, library, and folder allocations are being used. The capacity of your system (in megabytes) is displayed in the upper right-hand corner of the screen. There are 16 VTOC entries shown per screen; use the Roll keys to page through the entries. (Note: The bar graph is displayed using reverse-image screen attributes, which won't print when you press the Print key. Before requesting a screen print, press Command key 11 to fill the bars with printable characters. After printing, press Command key 12 to remove the printable characters.

If you want to restrict the display to particular files, libraries, or folders, use Command key 5 to display a selection prompt screen. (Figure 18-5 shows an example selection prompt screen.) Specify the VTOC entries whose allocations you want to examine by entering a complete or partial name in the NAME field, or leave the NAME field blank to display all entries. You can then select the type of entries you want displayed. The list of file types is self-explanatory (the default entry is Y; enter N for each type you do not want displayed). Press Command key 10 to display any alternate indexes associated with the file names you selected. To return to the bar graph screen, press the Enter key.

The bar-graph screen displayed initially (and redisplayed at any point with Command key 1) shows the percent of system space each selected file, library, or folder occupies, thereby clearly indicating which ones use the most space on your system. The highest value on the bar graph's reference scale reflects the highest percentage of disk space that one file, library, or folder occupies on the system. In Figure 18-4, the scale's highest value, 4 percent, is the highest percentage one file, library, or folder occupies on the system.

Two other bar-graph formats supply the rest of the information you need to monitor potential over- or under-allocation. When you use Command key 2, the bar graph shows you how much of a file, library, or folder's allocated space is being used. Using Command key 3, you can see the percent of space available. Together with the system-space percentage Command key 1 displays, you can effectively reallocate disk space to benefit system storage.

To locate over-allocations, use Command key 5 to select files, libraries, and folders. Then press Command key 3 to display available allocated space. Long bars on the graph indicate a significant amount of unused space, and you may want to decrease the allocation. Your space "savings" will be most significant with those entries (identified via the system-space bar graph) that occupy a large percent of system space. To find files, libraries, and folders with insufficient space, use Command key 5 to select files, libraries, and folders and Command key 2 to display the percentage of allocated space used. In this case, long bars on the graph indicate items that may need more space. Correcting both under- and over-allocations leads to more efficient disk use.

The VGRAPH utility is a useful programming weapon in your fight to allocate disk space more efficiently. Running VGRAPH regularly can help control wasted disk space — and help you win the "space battle."

Figure 18-1

Program VGRAPH

```
*        1         2         3         4         5         6         7         8
     H     064                      B       1                             VGRAPH
     F*
     F* Display an interactive bar graph of the disk VTOC
     F*  By Mel Beckman, 09/01/87
     F*
     F@WORKSTNCD  F    1898            WORKSTN
     F                                              KINFDS INFOS
     FREPORT  O   F     132     OF     PRINTER
     E*
     E* Screen arrays
     E*
     E                    LB       1100 12               File label
     E                    TP       1100  1               File type
     E                    RL       1100  4 0             Record length
     E                    RA       1100  8 0             Records allocated
     E                    RU       1100  8 0             Records used
     E                    CP       1100  8 0             Record capacity
     E*
     E* Dataset type selection mask arrays
     E*
     E                    SEL         8  1   SYN     1
     E*
     E* Bar graph arrays
     E*
     E                    LIN        16 80               Bar graph elements
     E                    BAR        51  1               Work array
     E*
     E* Scale messages
     E*
     E                    MSG     1   5 50
     E*
     E* Zero blanking work area
     E*
     E                    ZERO        8  1
     I/EJECT
     I*
     I* Bar graph screen
     I*
     I@WORKSTN         1 C1
     I                                        2   40SCALE
     I*
     I* Dataset selection screen
     I*
     I                 1 C2
     I                                        2   9 NAME
     I                                       10  17 SYN
     I/EJECT
     I*
```

```
I* SUBRVR is a routine to read the disk VTOC.
I*
I*
I* To read the VTOC
I*
I*           EXIT SUBRVR
I*           RLABL          NAME    8
I*           RLABL          VTOCDS
I*
I*
I* NAME  Contains the name of the file VTOC entry to read:
I*         If NAME is blank, then the next VTOC entry is read
I*         If NAME contains a file name, the VTOC entry for that file
I*         is read  (e.g  ARFILE)
I*         If NAME contains a partial file name, the next VTOC entry
I*         matching the partial name is read  A partial name is
I*         followed by an asterisk  (e.g  AR*)
I*         After reading a parent file, passing the keyword *ALTS in
I*         the name field will cause alternates for the parent to
I*         be retrieved on subsequent calls
I*         After reading to the end of the VTOC, a blank VTOC entry
I*         is returned  The next read will start at the beginning of
I*         the VTOC again (or at the point specified by the NAME field)
I*         The keyword *CONFIG passed in the name parameter returns
I*         some system configuration information in the VTOCDS
I*
I* VTOCDS  A data structure to contain the returned VTOC data
I*         This must be the name of a data structure to receive
I*         information about the file (record length, record count, etc)
I*         It must be at least 126 bytes long
I*
I* System configuration information returned from SUBRVR
I*
I*             From  To Name   Description
I*                1   2 CNREL  SSP release level
I*                3   4 CNMOD  SSP modification level
I*                5  21 CNBITS Configuration bits
I*                             (SCADSSPF through SCADCFGF, all
I*                             documented in SSP Data Areas)
I*               22  22 CPU#   System model
I*                             1-5364, 2-5360 3-5362
I*               23  28 CNDISK Disk capacity (in MB with two decimals)
I*
ICFIGDS      DS
I                                        1   20CNREL
I                                        3   40CNMOD
I                                        5  21 CNBITS
I                                       22  22 CPU#
I                                       23  282CNDISK
I*
I* VTOC data read via SUBRVR
I*
I*             From  To Name   Description
I*                1   1 FFORG  File organization
I*                             I-Indexed file      S-Sequential file
I*                             X-Alternate index   L-Library
I*                             F-Folder            D-Direct file
I*                2   2 FFLDAT Latest date indicator ('*')
I*                3  10 FFLABL File label
I*               11  16 FFCRDT Creation date
I*               17  17 FFTYPE File type
I*               18  18 FFSPIN Spindle number
I*               19  20 FFFLAG SSP flags
I*               21  26 FFATTR SSP attributes
I*               27  32 FFBLOK Block location
I*               33  40 FFRUSD Number of records used (if lib. #blocks)
I*               41  44 FFRECL Record length
I*               45  52 FFALOC Number of records or blocks alloc'd
I*               53  53 FFRORB 'R'- records allocated, 'B' - blocks
I*               54  56 FFKEYL Key length (for single part keys)
I*               57  60 FFKEYP Key position (for single part keys)
I*               61  68 FFCAPY File capacity in records
I*               69  76 FFXTND Extend value
I*               98 100 FFKYL1 Key length 1 (for multi-part keys)
I*              101 104 FFKYP1 Key position 1
I*              105 107 FFKYL2 Key length 2
```

```
I*              108 111 FFKYP2 Key position 2
I*              112 114 FFKYL3 Key length 3
I*              115 118 FFKYP3 Key position 3
I*              119 126 FFPARN Name of parent file
I*
IVTOCDS      DS
I                                         1   1 FFORG
I                                         2   2 FFLDAT
I                                         3  10 FFLABL
I                                        33  400FFRUSD
I                                        41  440FFRECL
I                                        53  53 FFRORB
I                                        45  520FFALOC
I                                        61  680FFCAPY
I                                       126 126 DUMMY
I/EJECT
I*
I* Bar data structure
I*
I            DS
I                                         1  80 BARDS
I                                         1   8 BLABL
I                                        10  10 BTYPE
I                                        11  18 BRUSED
I                                        19  26 BRAVL
I                                        28  28 BREV
I                                        29  79 BBAR
I/SPACE 3
I*
I* Zero blanking workarea
I*
I            DS
I                                         1   8 ZERO
I                                         1   8 ZERO8
I/SPACE 3
I*
I* Local data area contains initial selection parameters
I*
I           UDS
I                                       207 214 NAME
I                                       215 215 ALTFLG
I                                       216 223 SYN
I/SPACE 3
I*
I* INFDS data structure
I*
IINFDS       DS
I                                      *STATUS  STATUS
I/EJECT
C*
C* Initialization
C*
C*      Get the system configuration
C*      Compute system disk capacity in blocks
C*      Setup initial mode as "Percent of System"
C*    . Create various constants
C*
C                     MOVE '*CONFIG 'VNAME   8       Specify config get
C                     EXIT SUBRVR                    Read configuration
C                     RLABL          VNAME
C                     RLABL          CFIGDS
C*
C           CNDISK    MULT 1000000   WORK15 150      Compute
C           WORK15    DIV  2560      SYSSIZ  60       system size, blocks
C                     MOVE 'A'       MODE    1       Set default mode
C                     MOVE MSG,1     MESSAG           and default message
C*
C                     MOVEA'SDIXLFR?'SEL,1           Initialize typelist
C*
C                     BITOF'01234567'HEX00   1       Build hex constants
C                     MOVE HEX00     HEX20   1
C                     MOVE HEX00     HEX23   1
C                     MOVE HEX00     HEX31   1
C                     BITON'2'       HEX20   1         Normal field
C                     BITON'267'     HEX23   1         Rev Img High Int
C                     BITON'237'     HEX31   1         Rev Img Col Sep
```

```
C*
C* Main event loop
C*
C           EOJ       DOUEQ'Y'                        Until EOJ requested
C                     EXSR RDVTOC                      Read VTOC subfile
C   NU1               SORTAL8                          Sort if requested
C                     EXSR SHOW                        Display VTOC graph
C                     END                             Repeat
C*
C                     SETON                     LR
C/EJECT
C*
C* Read the VTOC into subfile arrays
C*
C           RDVTOC    BEGSR
C*
C                     MOVEL'99999999'L8               Fill with high value
C*                                                     (to speedup SORTA)
C                     Z-ADD0         MAXSIZ  60       Reset maximum size
C                     Z-ADD0         V       40       Reset VTOC index
C                     MOVE NAME      VNAME   8        Set VTOC name
C*
C           FFORG     DOULE*BLANK                     Until end of VTOC
C*
C                     EXIT SU8RVR                      Read a VTOC entry
C                     RLABL          VNAME   8
C                     RLABL          VTOCDS
C*
C           ALTFLG    IFEQ 'Y'                         If "show alts" set
C                     MOVE '*ALTS   'VNAME              Then set for alts
C                     MOVE ' '       ALTFLG             And reset flag
C                     END                              End IF
C*
C           FFORG     IFNE HEX00                       If not End of VTOC
C                     Z-ADD1         S
C           FFORG     LOKUPSEL,S                    11  Lookup type
C           SYN,S     IFEQ 'Y'                           If type selected
C                     ADD  1         V                    Bump index
C                     EXSR ADJUST                         Adjust values
C                     MOVELFFLABL    L8,V                 Save label
C                     MOVE V         LB,V                      index number
C                     MOVE FFORG     TP,V                      file type
C                     MOVE FFRECL    RL,V                      record length
C                     MOVE FFALOC    RA,V                      allocation
C                     MOVE FFRUSD    RU,V                      records used
C                     MOVE FFCAPY    CP,V                      capacity
C                     END                                End IF            E READING
C                     END                              End IF              E READING
C*
C                     END                             End DO               E READING
C                     ENDSR                                                E READING
C/EJECT
C*
C* Adjust dataset values depending on type of dataset
C*     Libraries are considered to have recl of 2560, so displayed size
C*     is rendered in blocks  The "records used" reflect the number of
C*     blocks used in the library for directory and active members
C*     Folders and alternate indexes get special treatment  The record
C*     count for an alternate index is always the same as the parent's
C*     record count, so we compute the allocated size in blocks
C*     The space available in a folder can't be easily determined, so
C*     folders appear to always be "full"
C*     The field MAXSIZ holds the size of the largest dataset passed
C*     through this routine, for use in autoscaling calculations
C*
C           ADJUST    BEGSR
C*
C           FFORG     IFEQ 'L'                          If library
C                     Z-ADDFFALOC    FFCAPY              FFCAPY-FFALOC
C                     END
C*
C           FFORG     IFEQ 'F'                          If folder
C                     Z-ADDFFALOC    FFRUSD              FFRUSD-FFALOC
C                     Z-ADDFFALOC    FFCAPY              FFCAPY-FFALOC
C                     END
C*
```

```
     C           FFORG     IFEQ 'X'                        If alternate
     C                     Z-ADDFFALOC    FFRUSD            FFRUSD=FFALOC
     C                     Z-ADDFFALOC    FFCAPY            FFCAPY=FFALOC
     C                     Z-ADD2560      FFRECL            FFRECL=2560
     C                     END
     C*
     C           FFCAPY    MULT FFRECL    WORK15 150       Compute maximum
     C           WORK15    DIV  2560      WORK15            file size for
     C                     ADD  1         WORK15            autoscaling
     C           WORK15    IFGT MAXSIZ
     C                     Z-ADDWORK15    MAXSIZ  60
     C                     END
     C*
     C                     ENDSR
     C/EJECT
     C*
     C* Display VTOC graph interactively
     C*
     C           SHOW      BEGSR
     C*
     C* If mode A, compute autoscale factor,
     C*  else use 100% scale.
     C*
     C           MODE      IFEQ 'A'
     C           MAXSIZ    DIV  SYSSIZ    WORK
     C           WORK      MULT 100       AUTOSC  30      11
     C   11                Z-ADD1         AUTOSC
     C                     Z-ADDAUTOSC    SCALE
     C                     ELSE
     C                     Z-ADD100       SCALE
     C                     END
     C*
     C* Loop to display screenfuls
     C*
     C                     Z-ADD1         X       30      Reset starting line
     C           EOJ       DOUEQ'Y'                       Do until EOJ
     C*
     C           SCALE     IFNE OLDSCL                     If scale changed
     C                     Z-ADDSCALE     OLDSCL  30        Then save old val
     C                     EXSR SCALE                       and make new one
     C                     END
     C*
     C                     EXSR PAGE                       Go build page
     C                     EXCPTSCRN1                      Display it
     C                     READ @WORKSTN              1111 Read the screen
     C*
     C* (Cmd6 to print all entries selected, added 05/10/88, DDS)
     C   KF                EXSR PRINT                      If Cmd6, print all
     C*
     C   KE                GOTO SELECT                     If Cmd5, do Select
     C*
     C   KG                MOVE 'Y'       EOJ     1        If Cmd7, set EOJ
     C   KK                MOVE '0'       PFILL   1        If Cmd11, set fill
     C   KL                MOVE ' '       PFILL            If Cmd12, clr fill
     C*
     C   KA                MOVE 'A'       MODE             If Cmd1
     C   KA                MOVE MSG,1     MESSAG 50         % of system used
     C   KA                Z-ADDAUTOSC    SCALE             use autoscale
     C*
     C   KB                MOVE 'B'       MODE             If Cmd2
     C   KB                MOVE MSG,2     MESSAG 50         % of records used
     C   KB                Z-ADD100       SCALE             force 100% scale
     C*
     C   KC                MOVE 'C'       MODE             If Cmd3
     C   KC                MOVE MSG,3     MESSAG 50         % of records avl
     C   KC                Z-ADD100       SCALE             force 100% scale
     C*
     C           STATUS    IFEQ 01122                      If roll-up
     C                     ADD  16        X                 then bump pointer
     C           X         IFGE V                           if X overflows
     C                     Z-ADD1         X                  then wrap around
     C                     END                              End IF
     C                     END                             End IF
     C*
     C           STATUS    IFEQ 01123                      If roll-down
     C                     SUB  16        X           1212  Then unbump X
```

```
C    12        V          SUB  16        X            1212   Adjust underflowTRY WRAPPING
C    12                   Z-ADD1         X                     If STILL too low THEN STAY PAGE-1
C                         END                                 End IF
C*
C                         END                                End DO              JUST REDISPLAY
C                         GOTO ENDSHW                                            JUST REDISPLAY
C*
C* Process dataset type selection screen
C*
C              SELECT     TAG
C                         EXCPTSCRN2                         Show select screen
C                         READ @WORKSTN               1111   Read screen
C    KG                   MOVE 'Y'       EOJ                 If Cmd7, set EOJ
C    KJ                   MOVE 'Y'       ALTFLG  1           If "show alts"
C   NKJ                   MOVE ' '       ALTFLG  1           If not "show alts"
C*
C              ENDSHW     ENDSR
C/EJECT
C*
C* Print full listing
C*
C              PRINT      BEGSR
C*
C                         TIME           TIME    60          Get time of day
C                         Z-ADD1         Y                   Set to line 1
C                         MOVE 'O'       PFILL               Set fill character
C                         EXCPTHEAD                          Print header
C*
C              1          DO   V         W                   Do V times
C                         EXSR BAR                            Build bar
C    OF                   EXCPTFOOT                           If ov, prnt footer
C    OF                   EXCPTHEAD                            and header
C                         EXCPTLINE                           Print line
C                         END                                End DO
C*
C                         EXCPTFOOT                          Print footer
C                         MOVE ' '       PFILL               Reset fill char
C*
C                         ENDSR
C/EJECT
C*
C* Build a new scale
C*
C              SCALE      BEGSR
C*
C              SCALE      IFEQ 0                             If scale too small
C                         Z-ADD001       SCALE                 Reset it
C                         END                                End IF
C              SCALE      IFGT 100                           If scale too large
C                         Z-ADD100       SCALE                 Reset it
C                         END                                End IF
C              SCALE      IFGE 100                           If scale is 100
C                         MOVE MSG,4     SCL     50           Then use default
C                         GOTO SCAEND                          and scram
C                         END
C*
C                         MOVE MSG,5     BAR                 Set bar
C              SCALE      DIV  50        WORK    85          Computer interval
C              WORK       MULT 5         INT     52
C                         Z-ADD0         VAL                 Save it
C*
C              4          DO   49        S       30          Do
C                         ADD  INT       VAL     52          Computer value
C                         MOVE VAL       ALPHA4  4           Make alpha
C                         MOVELALPHA4    ALPHA2  2
C                         MOVE ALPHA2    ALPHA1  1
C*
C              VAL        COMP 10                     11       IF LT 10
C    11                   MOVE ALPHA1    BAR,S                 THEN MOVE ONE DI
C                         MOVEAALPHA2    BAR,S                 ELSE MOVE TWO DI
C*
C                         END  5
C*
C                         MOVEABAR,1     SCL                   COPY SCALE
C              SCAEND     ENDSR
```

```
C/EJECT
C*
C* Build a bar graph page
C*
C           PAGE      BEGSR
C                     MOVE *BLANKS   LIN                Clear bar array
C                     Z-ADDX         W       40         Set starting point
C*
C                     DO   16        Y       20         Do 16 times
C           W         IFLE V                             If not end of array
C                     EXSR BAR                            Go build bar
C                     ADD  1         W                    Bump label index
C                     END                                End IF
C                     END                               End DO
C*
C           PAGEND    ENDSR
C/EJECT
C*
C* Build a bar
C*
C           BAR       BEGSR
C*
C* Build basic bar data structure
C*
C                     MOVE *BLANKS   BARDS               Clear bar DS
C                     MOVELLB,W      BLABL               Insert label
C                     MOVE LB,W      Z       40           (extract index)
C                     MOVE TP,Z      BTYPE               Insert type
C                     MOVE RU,Z      ZEROB               Zero blank
C                     EXSR ZEROBL                         records used
C                     MOVE ZEROB     BRUSED              and insert it
C           CP,Z      SUB  RU,Z      RAVL    80          Compute recs avail
C                     MOVE RAVL      ZERO8               Zero blank
C                     EXSR ZEROBL                         records avail
C                     MOVE ZERO8     BRAVL               and  nsert
C*
C* Set bar attribute to revimg for odd bars, revimg+colsep for even bars
C*
C                     MOVE Y         ONE      1          Set bar attribute
C                     TESTB'7'       ONE           11 Check even/odd
C  11                 MOVE HEX23     BREV                 If odd, revimg
C  N11                MOVE HEX31     BREV                 If even, +colsep
C*
C* Compute length of bar
C*
C           MODE      IFEQ 'A'                            If mode A
C           CP,Z      MULT RL,Z      WORK15 150            Figure blocksize
C           WORK15    DIV  2560      WORK15                of dataset
C                     ADD  1         WORK15                and compute
C           WORK15    DIV  SYSSIZ    WORK                  percent of system
C                     ELSE                                Else
C           MODE      IFEQ 'B'                             If mode B
C           RU,Z      DIV  CP,Z      WORK    85             compute % used
C                     ELSE                                 Else
C           MODE      IFEQ 'C'                              If mode C
C           RAVL      DIV  CP,Z      WORK                    compute % avail
C                     END                                   End IF
C                     END                                  End IF
C                     END                                 End IF
C*
C* Reduce percentage to array index
C*
C           WORK      MULT 100       WORK                Contrain to
C           100       DIV  SCALE     FACTOR  85          range of scale
C           WORK      MULT FACTOR    P       30H         and reduce
C                     DIV  2         P         H         to an index 1 to 51
C                     ADD  1         P             1111 If zero or negative
C  11                 Z-ADD1         P                    contrain to 1
C           P         COMP 51                   11       If > 51
C  11                 Z-ADD51        P                    constrain to 51
C*
C* Set length of bar by inserting screen attributes
C*
C                     MOVE PFILL     BAR                 Clear with fillchar
C                     MOVEA*BLANKS   BAR,P               Clear end of bar
C                     MOVE HEX20     BAR,P               Set end of bar
```

```
C                         MOVEABAR,1     BBAR                Insert in DS
C                         MOVE BARDS     LIN,Y               Put DS on screen
C*
C                         ENDSR
C/EJECT
C*
C* Blank leading zeros in array ZERO
C*
C              ZEROBL     BEGSR
C              1          DO   8         Q        20        For each digit
C              ZERO,Q     IFEQ '0'                           If zero
C                         MOVE *BLANK    ZERO,Q               Blank it
C                         ELSE                               Else
C                         Z-ADDB         Q                    Force exit
C                         END                                End IF
C                         END                               End DO
C                         ENDSR
C/EJECT
O*
O* Bar graph screen
O*
O@WORKSTNE                    SCRN1
O                                      K8  SCREENO1'
O                             SCALE     3
O                             CNDISK3  10
O                             MESSAG   60
O                             SCL     110
O                             LIN    1390
O                             SCL    1439
O*
O* Selection screen
O*
O         E                   SCRN2
O                                      K8 'SCREENO2'
O                             NAME      8
O                             SYN      16
O*
O* Print full graph listing
O*
OREPORT   E  105              HEAD
O                                       5 'Date''
O                             UDATE Y  14
O                                      48 'VTOC Bar Graph Listing'
O                                      57 'Time'
O                             TIME     66 '         '
O                                      78 'Capacity
O         E  1                HEAD
O                                       8 'Scale''
O                             SCALE    12
O                                      14 '%'
O                             CNDISK2  76
O                                      79 'MB'
O         E  1                HEAD
O                                      28 'Records or Blocks'
O                             MESSAG   78
O         E  1                HEAD
O                                      10 'Filename T'
O                                      18  Used'
O                                      26 'Avail
O                             SCL      78
O*
O         E  1                LINE
O                             LIN,1    80
O*
O         E  1                FOOT
O                             SCL      80
** Scale Messages
                 Percent of System
                   Percent Used
                 Percent Available
   10   20   30   40   50   60   70   80   90  100
```

Figure 18-2

Screen format member VGRAPHFM

```
*        1         2         3         4         5         6         7         8
0001 SSCREEN01    00      YY                                    23ABCEFGKL
     DFA0001     1 1 5Y Y          Y     Y            CT
     DFA0001    22 1 7Y                                C
0002 D        00220130Y               Y       Y        CVTOC BAR GRAPH DISPLAY
     DFA0002    17 153Y                                C
     DFA0001     9 171Y                                CCapacity
0007 DFA0002  00060202Y                                CScale
0007 DSCALE   00030209Y YN   B       Y
0007 DFA0004  00010213Y                                C%
     DFA0001    55 215Y                                C                       X
     D
     DFA0003     7 271Y
     DFA0002     2 279Y                                CMB
     DFA0002     9 3 2Y                                C
     DFA0001    17 312Y                                CRecords or Blocks
0003 0        00500330Y
0004 D        0008 4 2Y                       Y        CFilename
     DFA0005     1 411Y                       Y        CT
     DFA0004     7 413Y                       Y        C Used
     DFA0003     7 421Y                       Y        C Avail
0007 D        00500430Y
0009 D        12790502Y
0010 DSCALE   00502130Y
     DFA0002   24022 1Y                                CCmd1- % of system      X
     D                                                         Cmd2  % used    X
     D             Cmd5- Select files            Cmd11- Fill for printCmd3- % aX
     Dvailable           Roll Keys- paging              Cmd12- Turn off fill
0001 SSCREEN02            YY                                    GJ
     DFA0001     1 1 5Y Y          Y     Y            C2
     DFA0001    53 121Y               Y                CSelect datasets to dispX
     Dlay   Press ENTER to continue
0007 DFA0002  00060202Y                                C Name
     DFA0001     8 2 9Y Y                       Y
     DFA0002    42 339Y                                CYou can blank out NAME X
     Dto select all.
     DFA0004    42 439Y                                Cor enter a name (e g  AX
     DRFILE) or partial
     DFA0005    42 539Y                                Cname (e.g  AR*) to seleX
     Dct a subset   Press
     DFA0006    42 639Y                                CCmd10 to see alternatesX
     D for one parent
     DFA0001    33 7 2Y                                CDisplay Sequential fileX
     Ds?
     DFA0009     1 736Y YA                      Y
     DFA0002    33 8 2Y                                CDisplay Direct files?  X
     D
     DFA0010     1 836Y YA                      Y
     DFA0003    33 9 2Y                                CDisplay Indexed files? X
     0
     DFA0011     1 936Y YA                      Y
     DFA0004    3310 2Y                                CDisplay Alternate indexX
     Des?
     DFA0012     11036Y YA                      Y
     DFA0005    3311 2Y                                CDisplay Libraries?     X
     D
     DFA0013     11136Y YA                      Y
     DFA0006    3312 2Y                                CDisplay Folders?       X
     D
     DFA0014     11236Y YA                      Y
     DFA0007    3313 2Y                                CDisplay Remote files?  X
     D
     DFA0015     11336Y YA                      Y
     DFA0008    3314 2Y                                CDisplay other datasets?X
     D
     DFA0016     11436Y YA                      Y
     DFA0001   16023 1Y                                C   Cmd10-Display alternX
     Dates for specified parent                                                X
     D
```

Figure 18-3

Procedure VGRAPH

```
*
* Display VTOC bar graph
*
*  Paramter 1  Name or partial name to display   If blank, the entire VTOC
*                 is shown
*
*            2  'Y' - show alternates for the file matching parm1
*
*            3  'YYYYYYYY' is the type selection mask, as follows
*                 \\\\\\\_ Y-show unknown file types
*                  \\\\\\_ Y-show remote files
*                   \\\\\_ Y-show folders
*                    \\\\_ Y-show libraries
*                     \\\_ Y-show alternate indices
*                      \\_ Y-show indexed files
*                       \_ Y-show direct files
*                        \_ Y-show sequential files
*               This parameter defaults to 'YYYYYYYY' which shows everything
*               Use this parm to create "canned" VGRAPH procs for special cases
*
*            4  'NOSORT' means don't sort alphabetically
*
* Examples
*       VGRAPH CUSMAST,Y       (shows CUSMAST and all of its alternates)
*       VGRAPH TEST*,NNNNYNNN  (shows libraries named TESTxxxx)
*       VGRAPH                 (shows everything)
*
// IF ?4?/NOSORT SWITCH 10000000
// ELSE SWITCH 00000000
// LOCAL OFFSET-207,DATA-'?1?',BLANK-8   Name or partial name
// LOCAL OFFSET-215,DATA-'?2?',BLANK-1   'Y' to show alternates for parent
// LOCAL OFFSET-216,DATA-'?3'YYYYYYYY'?',BLANK-8 Type selection mask
// LOAD VGRAPH
// RUN
```

Figure 18-4

VTOC bar graph display

```
                              VTOC BAR GRAPH DISPLAY                  Capacity
 Scale  004 %                                                          60 02 MB
           Records or Blocks                     Percent of System
 Filename T  Used     Avail     00    00    01    01    02    02   .02    03    03    04
 JUNK1    I     921    2508  !!!!!
 JUNK2    I     921    2508  !!!!!
 MACROS   L      30      70  !!!!!!
 MELS3438 L     267     233  !!!!!!!!!!!!!!!!!!!!!!!!!!!
 MELTOOLS L      63      37  !!!!!!
 MEL2LIBR L     530     282  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 NAMEADDR S      12     107  !!
 NAMEPHON S      12      20  !!
 NEWS3438 L     292     208  !!!!!!!!!!!!!!!!!!!!!!!!!!!
 NTECH010 L     254     346  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 PROTOTYP L      59     241  !!!!!!!!!!!!!!!!
 SP0000   S     762       6  !!!
 SP0000X  S     255       1  !!
 SP0008   S     107      12  !!
 SP0009   S     105      14  !!
 SP0011   S      77       8  !!
                                00 . 00    01    01    02. .02    02 . 03    03.   04
Cmd1- % of system
Cmd2- % used                 Cmd5- Select files              Cmd11- Fill for print
Cmd3- % available        Roll Keys- paging                   Cmd12- Turn off fill
```

Figure 18-5
Selection screen

Re-creating Subroutine SUBRVR

If you don't have assembler subroutine SUBRVR, you can re-create it with procedure MKSUBRVR (you don't need IBM's Assembler Language Program Product to install SUBRVR). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRVR. You need to run MKSUBRVR only once because SUBRVR is subsequently linked into program VGRAPH when it is compiled.

```
// * Re-creating R-module SUBRVR in library #RPGLIB
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000031'     Number of #MAINT records
// LOCAL OFFSET-209,DATA-+
'[illegible]
// LOCAL OFFSET-273,DATA-+
'[illegible]
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-#MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRVR member to insert object code
// LOAD $FETIX
// RUN
HDR  38D4 SUBRV00000
PTF  2410 RSUBRVR 88 ,#RPGLIB
DATA C8A3 00 0000 [illegible]
DATA 5866 00 0020 [illegible]
DATA 0030 00 0040 [illegible]
DATA 4F12 00 0060 [illegible]
DATA C87C 00 0080 [illegible]
DATA 6488 00 00A0 [illegible]
DATA 182F 00 00C0 [illegible]
```

Continued

```
DATA BD60 00 00E0 7168031B71C08715563D7D185CC0821656C2011BCBC2021B1F0034302C282404
DATA FCB8 00 0100 E33314A10D071B661B9CF20122BC0000BC01010C07186F189178801BF2100D1C
DATA 92EA 00 0120 0718770BAC011A09AF010909F28753BC02000D071866189100333 11E14120503
DATA 6C98 00 0140 E33414D6F28147BC04000D071866186FF281380C07186F18668C01008C071118
DATA B119 00 0160 91C201185FE2020A7D5C00F281109C000000D20101E202017D00201C14120809
DATA 98BC 00 01B0 E336150D0000F10116C202181FBC00017D5C00F20103BA0401C20118CB84A105
DATA 4DC1 00 01A0 5F7F7F7FF401040B7D0002F2010D0C071B6F1891AF010909BC0001002F2D180B
DATA 4830 00 01C0 E3341542C087156BC20118CB7D8002F2013A7B801BF290340D0718661B9CF2B1
DATA 77C9 00 01E0 22C202181F2C1D185A1DBC0000BC00019C010954F401040B1C00242019170703
DATA C872 00 0200 E331157407187 70B0C1D1B3C1B5A35021B5EBC077D1B77C2A10000C2A20000F6
DATA 9DC1 00 0220 B0010E01156A18B1C087000034081 7E535011B5EC202002F2B23211200090702
DATA FBE4 00 0240 E33B15AD18CB5F7D7D7D6C09090BD201027D4000F202037C4B000201017D0000
DATA DDD7 00 0260 F101123601 1B5E57050F0F68020A156B030B1568020C166B030D166B02002201
DATA 28ED 00 0280 E33A15E80E1768030F176C0111197AF0116C01131B6C0014766C031B1F6C0019
DATA A81E 00 02A0 206C004C556C07545F571A6060680255606B0356606802576168035861 6B0200
DATA 1A8D 00 02C0 E336161F596268035A626802 5B636B035C6368025D6468035E6468025F656B03
DATA 54B8 00 02E0 60654C077D18917CD934B8803FF290037CC2342C021B1E238D02230000003222
DATA 59B1 00 0300 E33416541B94F284052C021B1E29C08717EE4C051F187E2C02181E388D0802F2
DATA 6B87 00 0320 012ABD5C03F20124C20118AB6C02152978070FF4014075010F0027161 20D0801
DATA 7B7A 00 0340 E33216B71C02181E0C1F02181E091E01181E1C3501185EC08717EE4C0727187F
DATA 8262 00 0360 8D0802F201108D5C03F2010A4C0627187E48602718842C00312C1B16120D0803
DATA AC05 00 0380 E33016B801161E3E3C001B1CC08717EE4C032B187F2C02181E413B80181CBD08
DATA A7F4 00 03A0 02F2010A4C032B18892C02181E2CC08717EE4C073300002D2B24191410 0B0702
DATA F8C8 00 03C0 E33116EA1B7FBD0802F201054C0633187EBD0042F2B12F2C00181E420F01181D
DATA 2AE1 00 03E0 1B1DC08717EE4C02371B7F2C01181E443C00181CC087002F2A26211D18160C01
DATA 5688 00 0400 E331171C17EE4C033B187F4630381B8557123B372C02181E4FC08717EE4C0743
DATA 1FA5 00 0420 1B7F2C021B1E52C08717EE4C074B187FB8801BF2909F002826211D18130B0601
DATA 23BE 00 0440 E330174DBD0057F2B12F2C001B1E570F011B1D181DC08717EE4C0263187F2C01
DATA 14C1 00 0460 181E593C001B1CC08717EE4C0367187F46306718850030 2B26221D1914100E09
DATA 63F6 00 0480 E331177F571 28763BD005AF2812F2C00181E5A0F01181D181DC08717EE4C026A
DATA C41E 00 04A0 1B7F2C011B1E5C3C00181CC08717EE4C036E1B7F4630002F2A26211D1B14120D
DATA AA74 00 04C0 E33017B06E188557126E6ABD005DF2812F2C00181E5D0F01181D181DC08717EE
DATA 3D0D 00 04E0 4C0271187F2C011B1E5F3C00181CC08717EE4C03750000 2D2924201B17151002
DATA CD25 00 0600 E33717E8187F46307518B5571275717D0000F28120C20217E5E202017E0000F1
DATA EB0B 00 0520 20096C0000007DC900F201097B8013F290037CE700C08700006FC6D900140601
DATA 6D1F 00 0540 E32F1818D3E5C4E2C93408181A07071B7F1B7F3C18181B0607187F187F0E0218
DATA C3F8 00 0560 1E181EF220043A01187F3F011818C00117FCC087002D29251E1C1816120E0C08
DATA 4192 00 0580 E302181B00000000000000000000000000000000000000000000000000000000
DATA 406A 00 05A0 0000000000000000000000000000000000000000000000000000000000000000
DATA 8144 00 05C0 E300186700000000000000000000000000000000000000000000000000000000
DATA D31B 00 05E0 0000000000000000000000000000000000000000000000000000000000000000
DATA 3E59 00 0600 E33B18B8000A0100F1F2F2F5F6F040404040404040400000005CC1D3E3E24040
DATA 95C9 00 0620 405CC3D6D5C6C9C740000800000A00C9C60000000000400000A100A080000000
DATA 5A74 00 0640 E31118CA19530000000000000000000000000000000000000000000000000000
DATA 0254 00 0660 0000000D00000000000000000000000000000000000000000000000000000001
DATA CC59 00 0680 C5FFFFCA00000000000000000000000000000000000000000000000000000000
DATA CE35 00 06A0 0000000000000000000000000000000000000000000000000000000000000000
DATA ABA1 00 06C0 615C000000000000000000000000000000000000000000000000000000000000
DATA C97E 00 06E0 0000000000000000000000000000000000000000000000000000000000000000
END  7BD5
```

Displaying Free Disk Space

by Gary T. Kratzer

program by Chuck Lundgren

Code on diskette:

Procedures VTOCFR, VTOCCM
RPG programs VTOCFR, VTOCCM
Screen format member VTOCFRFM
Assembler subroutine SUBRF5

The task of managing your disk space is a laborious one at best. The FILE and LIBR displays included in IBM's POP are great for showing what files

Figure 18-6

FREE screen organized by disk

```
VTOC       Free disk blocks:   6299 of   32098  Free VTOC entries:  229 of  360

Command keys   1:One column/disk  5:All disks  7:End  8:Location
              12:Compress

A1 - 040 Mb          A2 - 040 Mb
Location Blocks      Location Blocks
   8285      17         17723     388
   8328      29         18113      48
   9754      50         19635     200
  12146     854         19935     100
  13584     700         21012      50
  15074     460         21486     200
  15949     478         21705     250
                        22925     550
                        23558     600
                        24390      30
                        24500      50
                        26047     307
                      * 26467     160
                        26827     132
                        26970     250
                        27735      60
```

Figure 18-7

FREE screen organized by location

```
VTOC       Free disk blocks    6299 of   32098  Free VTOC entries   229 of  360

Command keys   1:One column/disk  5:All disks  7:End  8:Location  9:Size
              12:Compress

All Disks
Location Blocks      Location Blocks
   8285      17         24390      30
   8328      29         24500      50
   9754      50         26047     307
  12146     854         26467     160
  13584     700         26827     132
  15074     460         26970     250
  15949     478         27735      60
  17723     388
  18113      48
  19635     200
  19935     100
  21012      50
  21486     200
  21705     250
  22925     550
  23558     600
```

and libraries exist and how much space they occupy. But to see where your free disk space is, you must run a CATALOG and inspect the bottom portion of the printout. While this route isn't a bad way to go, it offers only one option for viewing the data. Wouldn't it be nice if POP also displayed free disk space on the screen so you wouldn't have to resort to the CATALOG option? Utility VTOCFR performs this function, and you can easily integrate it into your POP environment so that its operation appears seamless. VTOCFR is a utility that lists your disks' free space in a POP-like format (see Figures 18-6 and 18-7 for two versions of the FREE screen), with an additional option to run a "smart" COMPRESS in a variety of different ways.

The VTOCFR utility consists of RPG programs VTOCFR (Figure 18-8) and VTOCCM (Figure 18-9), screen format member VTOCFRFM (Figure 18-10), and procedures VTOCFR (Figure 18-11) and VTOCCM (Figure 18-12).

Procedure VTOCFR

When you call the VTOCFR procedure, you can specify up to three optional parameters. The first parameter determines which screen, FREE or COMPRESS, you want to see first. The second parameter determines how you want the free space displayed. Specify ONE to see the free space for each disk in separate columns, or ALL to display the free space continuously in all columns. (Note that the second parameter applies only if parameter 1 is FREE.) You use parameter 3 to specify whether to display the free space by LOCATION or SIZE. (Note that SIZE is valid only if you specify ALL in parameter 2.)

When calling procedure VTOCFR, simply type in the parameters you want to specify. If you enter no parameters, the defaults are FREE, ALL, and LOCATION, respectively. After you press Enter, either the FREE screen or the COMPRESS screen is displayed.

The FREE Screen

Notice that the "look and feel" of the FREE screen is similar to POP's FILE display. The top line displays the number of free disk blocks and free VTOC entries in relation to the total number existing of each. Farther down the screen are columns of 16 entries each that display the free space beginning-block locations and the total number of blocks at each location.

Specifying FREE, ONE, and LOCATION for your initial parameters displays a FREE screen like the one in Figure 18-6. If you choose FREE, ALL, and LOCATION as the initial parameters, a screen like Figure 18-7 is the result. By pressing the command keys listed near the top of the FREE screen, you can view the free disk space in one of three ways. First, you can press Command key 1 to display information about each disk separately. This option (Figure 18-6) displays free space for spindle A1 in column 1, A2 in column 2, and so on. If more than 16 "holes" exist for each disk, press Enter to display additional entries. Note that in this mode, the columns scroll in unison by spindle. In other words, in our example figure, spindle A1 has only seven holes, while A2 has at least 16 holes. If you press Enter, you see up to 16 more entries for A2, but the first column is blank because the screen has already displayed all entries for A1.

You can also display the free disk space in two other ways — as a continuous list arranged by location, or as one arranged by size. To do this, you press Command key 5. A screen like that shown in Figure 18-7 appears, listing free space according to location. To view the free space by size, you press Command key 9. These two options disregard which spindle the free space resides on; rather, they treat all spindles as one disk and display the entries in the format you have requested.

COMPRESS Options

In addition to displaying the free space with utility VTOCFR, you have the option of running a compress of your disks. You can do this either by specifying COMPRESS for parameter 1 when you call procedure VTOCFR or by pressing Command key 12 on the FREE screen. Procedure VTOCFR then calls procedure VTOCCM, which displays the VTOCFR COMPRESS screen (Figure 18-13). The first prompt asks whether you want to perform a COMPRESS ALL or to compress each disk individually. If you choose ALL, the procedure either invokes or JOBQs a COMPRESS ALL, depending on whether you pressed Enter or Command key 4. A COMPRESS ALL compresses the disks differently, depending on how many spindles you have. (Refer to the *System Reference* manual (SC21-9020) for more detailed information.) If you specify EACH in response to the first prompt, you must then press Enter and indicate, according to further prompts, which disks you want compressed and where the free space should be collected (FREEHIGH or FREELOW). You are prompted only for the disks installed on your machine. For instance, if you have just two spindles, you will be prompted for A1 and A2 only. Once you've made your decision, press Enter to perform the COMPRESS immediately, or press Command key 4 to JOBQ it.

Easy POP Integration

You can easily integrate utility VTOCFR directly into POP by modifying two procedures in #POPLIB. In both the FILE@ and LIBR@ procedures, look for the word COMPRESS, and replace it with VTOCFR. Then copy the VTOCFR modules to #POPLIB. Now, utility VTOCFR can help you quickly spot where your free space exists: whenever you're viewing the POP FILE or LIBR list displays, you can press Command key 12 to invoke VTOCFR (instead of COMPRESS). You can then make an informed decision about whether a compress is really necessary and, if so, how to perform it. So throw away those catalog listings, and put utility VTOCFR to work for you!

Figure 18-8

Program VTOCFR

```
*.. .  1 .  ... 2 .   ... 3  .. ... 4  ..  .. 5 ...   6 .... . 7 . .  .. 8
     H    064                          B      1                        VTOCFR
     F*****************************************************************
     F* PROGRAM NAME.     VTOCFR
     F* DESCRIPTION.   .. POP-like display of free space on all disks.
     F* PROGRAMMER...     Chuck Lundgren (Iris Software, Inc.)
     F*   (c) COPYRIGHT 1990 Iris Software, Inc  - All Rights Reserved
     F* DATE WRITTEN   .  January 1990
     F*
     F* N40 Display message on screen: "No more than xxx free blocks...".
     F*  41 Highlight "1:One column/disk" on screen.
     F*  42 Highlight "5:All disks" on screen
     F*  43 Highlight "8:Location" on screen
     F*  44 Highlight "9:Size" on screen.
     F* N47 Display disk name for A2.
     F* N48 Display disk name for A3.
```

```
F* N49 Display disk name for A4
F* N50 Display "9-Size" on screen
F* N57 Display column heading for disk A2
F* N58 Display column heading for disk A3
F* N59 Display column heading for disk A4
F*
F* VERSION DATE       FIX DESCRIPTION
F*---------------------------------------
F*************************************************************************
FWORKSTN CD  F     820              WORKSTN
F                                                   KFMTS  VTOCFRFM
E*************************************************************************
E                   MBK      1  11  6 OAMMB      3 0  Dsk model-Blocks & Mb
E                   DSS          5  6                 Start sect  disk adr.
E*                                                      DSS,1 - Not used
E*                                                      DSS,2 - Disk A2
E*                                                      DSS,3 - Disk A3
E*                                                      DSS,4 - Disk A4
E*                                                      DSS,5 - Always zero
E*                                                       (Stops loop)
E                   DEB      4   4  6 OA              End block address
E                   DMB          4  3 0               Disk sizes in Mb.
E                   HEX     16  16  1   DEC      2 0  Hex/decimal conv
E                   HEXV         6  1                 Hex digits
E*
E* Note  If you increase the no  of elements in ADSZ, D and SZIX, then
E* you need to set FREMAX to the new no  of elements in subr  FIRST
E*
E                   ADSZ       500 12                 Addr/size by address.
E                   D          500  1 D               Disk by address
E*                                                      Corresponds to each
E*                                                      element in ADSZ.
E                   SZIX       500 10  D              Size/index by size
E*                                                      1-6  Size
E*                                                      8-10 ADSZ pointer.
E                   DP           4  4 0               Disk pointer in D
E                   SCAD        64  6 D               Screen block addrs
E                   SCSZ        64  6 0               Screen block sizes
I*************************************************************************
IWORKSTN          1 C1
I*
I*-----------------------------------
I           DS
I                                        1  10 DSSZIX
I                                        1   60DSSIZE
I                                        7  100DSINDX
I           DS
I                                        1  11 COLHED
I                                        6   80DSAIMB
I*-----------------------------------
I           UDS
I                                      171 178 DPER
I                                      179 181 SHOW
I                                      182 189 SORT
I                                      214 2140DT
C*************************************************************************
C*
C                     EXSR FIRST                        Do first time
C           END       DOWNE'Y'                          Do while not eoj
C                     EXSR DSPF                           Disp  free blocks
C                     END                               End DO
C                     SETON                     LR
C*
C*-----------------------------------
C* Calculate size of each disk
C*
C           CALDS     BEGSR
C*
C                     Z-ADDO         BEGBLK  60         A1 start is zero.
C                     Z-ADD999999    DEB                Set high for LOKUP.
C*
C           1         DO   4         L                  Do for each disk
C           L         ADD  1         J                    Point to next dsk
C           DSS,J     IFNE '000000'                       If not last disk
C                     MOVEADSS,J     HEXV                   Save for conv.
C                     EXSR HX2DC                            Hex->decimal
```

```
C          DECVAL    DIV  10        ENDBLK  80             Sectors->blocks
C          ENDBLK    SUB  BEGBLK    DSKSIZ  60             Disk size.
C          ENDBLK    SUB  1         DEB,L                  Save end block
C                    Z-ADDENDBLK    BEGBLK                 Next disk.
C                    ELSE                                Else last disk
C          BTTL      SUB  BEGBLK    DSKSIZ                 Disk size
C                    ADD  BBEG      DSKSIZ                 Adj. w/beg.blk
C          BTTL      ADD  BBEG      DEB,L                  Save end block
C                    Z-ADDL         DT      10             Save ttl  dsks
C                    END                                 End IF
C*
C                    Z-ADD1         K       20           Reset index
C          DSKSIZ    LOKUPMBK,K                     2020 Find disk model
C   20               Z-ADDMMB,K     DMB,L                Get disk size
C*
C          DSS,J     IFEQ '000000'                       If last disk
C                    Z-ADD4         L                      Stop loop
C                    END                                 End IF
C                    END                               End DO
C*
C                    ENDSR
C*
C*-----------------------------------
C* Display free blocks and read screen
C*
C          DRF '     BEGSR
C*
C* If last screenful of blocks was displayed, fetch it over again
C*
C          LASTSC    IFEQ 'Y'                          If last screen
C                    MOVE 'Y'       UPDTSC               update it
C                    END                               End IF
C*
C* If cmd key pressed, or this is the first time through, or the
C* last screenful of blocks is currently displayed, then get and process
C* Format-5s
C*
C          UPDTSC    IFEQ 'Y'                          If update request
C                    EXSR GETF5                          Get F5s
C                    EXSR SETAR                          Setup arrays
C          ADSZ,1    IFEQ *BLANK                         If no free blocks
C                    Z-ADDO         FREBLK  40             flag it
C                    ELSE                                Else
C                    EXSR SRTAR                            sort arrays
C                    END                                 End IF
C                    END                               End IF
C*
C* Display a screenful of free blocks, then read screen
C*
C                    MOVE 'N'       LASTSC  1          Assume not last scn
C          SHOW      IFEQ 'ALL'                        If Cmd5:Show all
C                    EXSR DSPF5                          display a screen
C                    ELSE                              Else Cmd1 1 dsk/col
C                    EXSR DSPF1                          display a screen
C                    END                               End IF
C*
C                    EXCPTFREE                         Display and
C                    READ WORKSTN                   3030  read the screen
C                    MOVE 'N'       UPDTSC             Reset update flag
C*
C                    ENDSR
C*
C*-----------------------------------
C* Display screen showing all free system blocks
C*
C          DSPF      BEGSR
C                    EXSR DRF                            Display, read scn
C   KA               EXSR F01                            Display 1 col/dsk
C   KE               EXSR F05                            Display all disks
C   KG               EXSR F07                            End job
C   KH               EXSR F08                            Sort by location
C   KI               EXSR F09                            Sort by size
C   KL               EXSR F12                            Compress
C                    ENDSR
C*
C*-----------------------------------
```

```
C* Display a screenful of free blocks in Cmd1:One column/disk mode
C*
C          DSPF1     BEGSR
C*
C                    Z-ADD0         SCAD                Clear screen addr.
C                    Z-ADD0         SCSZ                  and size arrays.
C*
C          1         DO   4         DN        10        Do for each disk
C          DP,DN     IFGT 0                               If disk has blks
C          DN        SUB  1         S                       Point to top
C                    MULT 16        S                       row of
C                    ADD  1         S                       the column
C*
C          1         DO   16        L         40            Do for @ row
C                    Z-ADDDP,DN     AP                        Get blk ptr
C                    Z-ADDD,AP      DNUM      10              Get disk no
C          DN        IFEQ DNUM                                If disk match
C                    MOVELADSZ,AP   SCAD,S                      get address
C                    MOVE ADSZ,AP   SCSZ,S                      and size
C*
C          BC        IFEQ FREBLK                                If last blk
C                    Z-ADD16        L                             end both
C                    Z-ADD4         DN                            loops, &
C                    MOVE 'Y'       LASTSC                        last scn
C                    ELSE                                       Else
C                    ADD  1         S                             Inc  ptr
C                    ADD  1         DP,DN                         Inc  ptr
C                    ADD  1         BC                            Inc  ctr
C                    END                                        End IF
C*
C                    ELSE                                     Else
C                    Z-ADD16        L                           no more row
C                    END                                      End IF
C*
C                    END                                    End DO
C                    END                                  End IF
C                    END                                End DO
C*
C                    ENDSR
C*
C*-----------------------------------
C* Display a screenful of free blocks in Cmd5:Show all mode
C*
C          DSPF5     BEGSR
C*
C                    SETON                      474849  Turn off disk names
C                    SETOF                      575859  Turn on col  hdgs
C                    Z-ADD0         SCAD                Clear screen addr
C                    Z-ADD0         SCSZ                and size arrays
C*
C          1         DO   64        S         20        Do for each pos
C*
C          SORTBY    IFEQ 'L'                             If sort by loc
C                    MOVELADSZ,BC   SCAD,S                  fetch address
C                    MOVE ADSZ,BC   SCSZ,S                  and size
C                    ELSE                                 Else size sort
C                    MOVE SZIX,BC   AP        40            Get index
C          AP        IFEQ 0                                 If no more
C                    Z-ADD64        S                         end loop
C                    MOVE 'Y'       LASTSC                    & last scrn
C                    ELSE                                   Else
C                    MOVELADSZ,AP   SCAD,S                    get address
C                    MOVELSZIX,BC   SCSZ,S                    and size
C                    END                                    End IF
C                    END                                  End IF
C*
C          BC        IFEQ FREBLK                          If last block
C                    Z-ADD64        S                       end loop
C                    MOVE 'Y'       LASTSC                  & last scrn
C                    ELSE                                 Else
C                    ADD  1         BC                      next-block.
C                    END                                  End IF
C                    END                                End DO
C*
C                    ENDSR
C*
```

```
C*-----------------------------------
C* Free blocks screen: Cmd1 - display one disk in each column.
C*
C          F01       BEGSR
C                    MOVE 'ONE'     SHOW               Show one disk/col.
C                    MOVEL'A1 - '   COLHED             Build column sub-
C                    MOVE '    Mb'  COLHED             heading for disk
C                    Z-ADDDMB,1     DSA1MB             no. 1.
C                    MOVE 'Y'       UPDTSC  1          Update the screen.
C                    SETON                    414350 HI Cmd1/8;INH  Cmd9
C                    SETOF                    42     Dim Cmd5.
C                    ENDSR
C*
C*-----------------------------------
C* Free blocks screen: Cmd5 - display all disks beginning in column 1.
C*
C          F05       BEGSR
C                    MOVE 'ALL'     SHOW               Show all disks
C                    MOVEL'All '    COLHED               then note it on
C                    MOVE 'Disks  ' COLHED               screen.
C                    MOVE 'Y'       UPDTSC             Update the screen.
C                    SETON                    42     Highlight Cmd5
C                    SETOF                    41  50 Dim Cmd5 & dsp Cmd9
C          SORTBY    COMP 'L'                 444443 HI Cmd8 or Cmd9
C                    ENDSR
C*
C*-----------------------------------
C* Free blocks screen: Cmd7 - exit the program.
C*
C          F07       BEGSR
C                    MOVE 'Y'       END     1          End of job
C                    ENDSR
C*
C*-----------------------------------
C* Free blocks screen. Cmd8 - select sort by location.
C*
C          F08       BEGSR
C                    SETOF                    44     Dim Cmd9.
C                    SETON                    43     Highlight Cmd8.
C                    MOVE 'Y'       UPDTSC             Update screen.
C                    MOVE 'L'       SORTBY  1          Sort by location.
C                    ENDSR
C*
C*-----------------------------------
C* Free blocks screen: Cmd9 - select sort by size.
C*
C          F09       BEGSR
C          SHOW      IFEQ 'ALL'                        If Cmd5:Show all
C                    SETOF                    43       Dim Cmd8.
C                    SETON                    44       Highlight Cmd9.
C                    MOVE 'S'       SORTBY               sort by size,
C                    MOVE 'Y'       UPDTSC               update screen,
C                    END                               End IF
C                    ENDSR
C*
C*-----------------------------------
C* Free blocks screen: Cmd12 - compress the disk
C*
C          F12       BEGSR
C                    MOVE 'Y'       END     1          End of job
C                    MOVE 'COMPRESS'OPER               Select compress.
C                    ENDSR
C*
C*-----------------------------------
C* Initialize variables.
C*
C          FIRST     BEGSR
C                    Z-ADD500       FREMAX  40         Max  free blocks.
C                    EXSR GETF5                        Get Format-5s
C                    EXSR CALDS                        Calc. disk sizes.
C          OPER      IFEQ 'FREE    '                   If display space
C          SHOW      CASEQ'ONE'     F01                  Fake press Cmd1
C                    CAS            F05                            ..Cmd5
C                    END                                 End CASE
C          SORT      CASEQ'LOCATION'F08                  Fake press Cmd8
C                    CAS            F09                            ..Cmd9
```

```
C                  END                                  End CASE
C                  ELSE                                Else compress
C                  EXSR F12                              Fake press Cmd12
C                  END                                 End IF
C                  ENDSR
C*
C*-----------------------------------
C* Get the Format-5s.
C*
C          GETF5     BEGSR
C                    MOVE '000000'  DSS,5               Initialize.
C                    EXIT SUBRF5                        Get free space.
C                    RLABL          ADSZ                Block addr/size
C                    RLABL          BTTL    60          Total blocks
C                    RLABL          BBEG    60          Beg. user blocks.
C                    RLABL          DSS,2               A2 start sector.
C                    RLABL          DSS,3               A3 start sector
C                    RLABL          DSS,4               A4 start sector.
C                    RLABL          VTTL    40          Total VTOC entries
C                    RLABL          VUSD    40          Used VTOC entries.
C                    RLABL          RET     10          Return code
C          RET       COMP 3                    4040     Enough array space?
C*
C                    EXSR MSKDS                         Mask unused disks.
C          VTTL      SUB  VUSD      VAVL    40          Avail. VTOC entries
C                    ENDSR                              N-Display message.
C*
C*-----------------------------------
C* Convert hex to decimal.
C*
C          HX2DC     BEGSR
C                    Z-ADD1         DIGM    80          Digit multiplier.
C                    Z-ADDO         DECVAL  80          Decimal equivalent.
C                    Z-ADD6         H       10          Point to LS digit.
C          H         DOWGE1                             Do for @ hex digit.
C                    Z-ADD1         DI      20            Reset digit indx.
C          HEXV,H    LOKUPHEX,DI                    20    Get dec. value.
C          DEC,DI    MULT DIGM      TEMP80  80            Actual value.
C                    ADD  TEMP80    DECVAL                Accumulate.
C                    MULT 16        DIGM                  Bump multiplier.
C                    SUB  1         H                     Next digit.
C                    END                                End DO.
C                    ENDSR
C*
C*-----------------------------------
C* Mask unused disk from display on screens
C*
C          MSKDS     BEGSR
C          DSS,2     COMP '000000'                  47 No A2 disk info
C          DSS,2     COMP '000000'                  57 No A2 col. heading.
C          DSS,3     COMP '000000'                  48 ...A3..
C          DSS,3     COMP '000000'                  58  ..A3...
C          DSS,4     COMP '000000'                  49 ...A4.
C          DSS,4     COMP '000000'                  59 .  A4..
C                    ENDSR
C*
C*-----------------------------------
C* Set up arrays containing sizes and disk numbers for each free block.
C*
C          SETAR     BEGSR
C*
C                    Z-ADDFREMAX    FREBLK  40            Assume max. blks.
C                    MOVE *BLANKS   SZIX                  Clear size/index.
C                    Z-ADDO         D                     Clear disk #.
C                    Z-ADDO         BAVL    60            Clear free blks.
C                    Z-ADD1         J                     Point to first.
C*
C          J         DOUEQFREMAX                          Do for all blocks
C          ADSZ,J    IFNE *BLANKS                           If valid block
C                    MOVELADSZ,J    ADDR    60                Get address.
C                    MOVE ADSZ,J    DSSIZE                    Get size.
C                    ADD  DSSIZE    BAVL                      Accumulate.
C*
C          SHOW      IFEQ 'ONE'                               If 1 dsk/col.
C                    Z-ADD1         L                           Reset disk#
C          ADDR      LOKUPDEB,L                20  20           Get disk #.
```

```
     C                   Z-ADDL         D,J                          Save disk#.
     C                   END                                       End IF
     C*
     C           SORTBY  IFEQ 'S'                                  If size sort
     C                   Z-ADDJ         DSINDX                       ASDZ index.
     C                   MOVE DSSZIX    SZIX,J                       Save elem
     C                   END                                       End IF
     C*
     C                   ADD  1         J                          Next block
     C*
     C                   ELSE                                    Else last block
     C           J       SUB  1         FREBLK                     save # blocks
     C                   Z-ADDFREMAX    J                          and end loop.
     C                   END                                     End IF
     C                   END                                   End DO
     C*
     C                   ENDSR
     C*
     C*----------------------------------
     C* Sort free blocks in the sequence requested by user.
     C*
     C           SRTAR   BEGSR
     C*
     C* Sort by size (if Cmd9:Size is selected)
     C*
     C           SORTBY  IFEQ 'S'                              If sort by size
     C                   SORTASZIX                               then sort it
     C                   END                                   End IF
     C*
     C                   Z-ADD1         BC      40             Reset block cntr.
     C*
     C*    and set pointers to 1st free block on each disk (if Cmd1 selected)
     C*           .
     C           SHOW    IFEQ 'ONE'                            If 1 disk/column
     C                   Z-ADDO         DP                       reset pointers
     C           1       DO   4         DN                       Do for @ disk.
     C                   Z-ADD1         J       40                 Reset ptr
     C           DN      LOKUPD,J                     20           Find 1st blk
     C   20              Z-ADDJ         DP,DN                      Point to it
     C                   END            .                        End DO
     C                   END                                   End IF
     C*
     C                   ENDSR
     O*----------------------------------
     OWORKSTN E                  FREE
     O                                     KB 'VTOCFR01'
     O                           BAVL  Z    6
     O                                      9 'of'
     O                           BTTL  Z   16
     O                           VAVL  Z   20
     O                                     23 'of'
     O                           VTTL  Z   28
     O                           FREMAXZ   32
     O                           COLHED    43
     O                           DMB,2     46
     O                           DMB,3     49
     O                           DMB,4     52
     O                           SCAD     436 '      '
     O                           SCSZ     820 '      '
** MBK MM8  Disk Model  Array index
 12049 30   21ED-A2       1
 16374 40   HD53          2
 24098 60   21ED-A3       3
 26390 65   0667          4
 41496105   0669          5
 78204200   10SR-1        6
 78236200   9332-A11      7
122811315   ????          8
140218358   10SR-2        9
156472400   9332-A12     10
999999999   None         11
** DEB

** HEX/DEC
000101202303404505606707808909A10B11C12D13E14F15
```

Figure 18-9

Program VTOCCM

```
*   .  1    .   2 .      3      . . 4     . . 5 .      6       . 7   .  . . 8
H     064                           B       1                             VTOCCM
F***********************************************************************
F* PROGRAM NAME      VTOCCM
F* DESCRIPTION       Smart compress prompt screen.  VTOCCM uses
F*    information from VTOCFR to know how many disks there are on the S/36
F* PROGRAMMER.       Chuck Lundgren (Iris Software, Inc.)
F*    (c) COPYRIGHT 1990 Iris Software, Inc  - All Rights Reserved
F* DATE WRITTEN      January 1990
F*
F*  45 Compress each disk individually (protect ALL/EACH field on screen).
F* N46 Display COMPRESS prompts for disk A1.
F* N47 Display COMPRESS and column heading for disk A2
F* N48 Display COMPRESS and column heading for disk A3
F* N49 Display COMPRESS and column heading for disk A4
F*  51 Error for ALL/EACH compress parameter
F*  52 Error for A1 compress parameters.
F*  53 Error for A2 compress parameters.
F*  54 Error for A3 compress parameters.
F*  55 Error for A4 compress parameters.
F*  56 Position cursor on disk A1 in compress prompt screen.
F*
F* VERSION DATE      FIX DESCRIPTION
F* ------------------------------------
F***********************************************************************
FWORKSTN CD  F     820            WORKSTN
F                                                 KFMTS  VTOCFRFM
E***********************************************************************
E                    SCD        4  1                   Screen compress Y/N.
E                    SLH        4  4                   Screen FREELOW/HIGH.
E                    CDLH       4  5                   Compress Y/N & LOW/HI
E                    CMD     1  2 60                   Command key line.
I***********************************************************************
IWORKSTN          1 C2
I                                          2   5 SALLEA
I                                          6   6 SCD,1
I                                          7  10 SLH,1
I                                         11  11 SCD,2
I                                         12  15 SLH,2
I                                         16  16 SCD,3
I                                         17  20 SLH,3
I                                         21  21 SCD,4
I                                         22  25 SLH,4
I*-----------------------------------
I          UDS
I                                        190 193 ALLEA
I                                        194 213 CDLH
I                                        214 2140DT
I                                        215 215 ACTION
C***********************************************************************
C*
C                    EXSR FIRST                          Do first time.
C          END       DOUEQ'Y'                            Do until eoj
C          CPRMPT    CASEQ'1'       DSPC1                  Display 1st half.
C                    CAS            DSPC2                  Display 2nd half.
C                    END                                   End CASE
C                    END                                 End DO
C                    SETON                     LR
C*
C*-----------------------------------
C* Compress screen: Cmd2 - page back
C*
C          CO2       BEGSR
C                    MOVE '1'       CPRMPT               Display prompt 1
C                    ENDSR
C*
C*-----------------------------------
C* Compress screen  Cmd4 - put on job queue
C*
C          CO4       BEGSR
C                    MOVE 'J'       ACTION  1            Put on jobqueue.
C                    ENDSR
C*
C*-----------------------------------
C* Compress screen. Cmd7 - cancel compress.
C*
```

```
C          C07       BEGSR
C                    MOVE 'Y'       END     1         End program
C                    MOVE 'E'       ACTION            End proc/no compr
C                    ENDSR
C*
C*-----------------------------------
C* Compress screen  Enter pressed - save compress parameters
C*
C          CSAV      BEGSR
C                    MOVE SALLEA    ALLEA             Save ALL/EACH parm
C                    MOVELSCD       CDLH              Save compress parm
C                    MOVE SLH       CDLH              Save LO/HI parms
C                    MOVE 'Y'       END               End program
C                    MOVE 'C'       ACTION            Do compress
C                    ENDSR
C*
C*-----------------------------------
C* Display 1st prompt for compressing all or individual disks
C*
C          DSPC1     BEGSR
C                    SETOF                     45  56 Unprot  & unpos
C                    SETON                     464748 Protect fields
C                    SETON                     49
C                    MOVE CMD,1     CMDLIN 60         Get cmd key line.
C          ERROR     DOUEQ*BLANK                      Do until no error
C                    MOVE ' '       ERROR               Assume no error
C                    EXCPTCOMP                          Display prompt
C                    READ WORKSTN                 3030  Read screen
C   KG               EXSR C07                           Cancel compress
C*
C          END       IFNE 'Y'                           If not cancelled
C          SALLEA    IFEQ 'ALL '                          If compress all
C                    EXSR CSAV                              Save parms
C   KD               EXSR C04                               Put on jobq
C                    END                                  End IF
C*
C          SALLEA    IFEQ 'EACH'                          If select disks
C                    MOVE '2'       CPRMPT                  select disks
C                    END                                  End IF
C*
C          SALLEA    COMP 'ALL '               5151       Not 'ALL'
C   51     SALLEA    COMP 'EACH'               5151       and not 'EACH'
C   51               MOVE 'Y'       ERROR   1
C                    END                                End IF
C                    END                              End DO
C                    ENDSR
C*
C*-----------------------------------
C* Display 2nd prompt for selecting disks to compress
C*
C          DSPC2     BEGSR
C                    SETON                     45  56 Pro., pos  cursor
C                    SETOF                     46     Unprotect A1 prmpt
C                    EXSR MSKDS                       Mask unused disks
C                    MOVE CMD,2     CMDLIN 60         Get cmd key line
C          ERROR     DOUEQ*BLANK                      Do until no error
C                    MOVE ' '       ERROR               Assume no error
C                    EXCPTCOMP                          Display prompt
C                    READ WORKSTN                 3030  Read screen
C   KB               EXSR C02                           Previous prompt
C   KG               EXSR C07                           Cancel compress
C*
C  NKB     END       IFNE 'Y'                           If not Cmd2 or 7
C                    MOVE 'N'       DSKSEL  1             Assume none
C          1         DO   DT        L      10             Do for @ disk
C          SCD,L     IFEQ 'Y'                               If selected
C                    MOVE 'Y'       DSKSEL                    note it
C                    END                                    End IF
C                    END                                  End DO
C          DSKSEL    IFEQ 'N'                             If no selection
C                    EXSR C07                               treat as Cmd7
C                    END                                  End IF
C                    END                                End IF
C*
C  NKB     END       IFNE 'Y'                           If not Cmd2 or 7
C          1         DO   DT        L                     Do for @ disk
```

```
     C           SCD,L     COMP 'Y'                 2020        Not 'Y',
     C   20      SCD,L     COMP 'N'                 2020        not 'N'
     C           SLH,L     COMP 'LOW '              2121        not 'LOW', &
     C   21      SLH,L     COMP 'HIGH'              2121        not 'HIGH'.
     C*
     C   20
     COR 21      1         DO   1                               Do for error
     C                     MOVE 'Y'       ERROR                   Flag error
     C           L         COMP 1                     52          for A1
     C           L         COMP 2                     53              A2
     C           L         COMP 3                     54            . A3
     C           L         COMP 4                     55              A4
     C                     Z-ADDDT        L                       End loop
     C                     END                                  End DO
     C                     END                                End DO
     C*
     C           ERROR     IFEQ *BLANK                        If no errors
     C                     EXSR CSAV                            Save parms
     C   KD                EXSR C04                             Put on jobq
     C                     END                                End IF
     C                     END                              End IF
     C                     END                            End DO
     C                     ENDSR
     C*
     C*-----------------------------------
     C* Initialize variables.
     C*
     C           FIRST     BEGSR
     C                     MOVE ALLEA     SALLEA          Set ALL/EACH parm
     C                     MOVELCDLH      SCD             Set Y/N parms
     C                     MOVE CDLH      SLH             Set LO/HI parms
     C                     MOVE '1'       CPRMPT  1       Display prompt 1
     C                     ENDSR
     C*
     C*-----------------------------------
     C* Mask unused disk from display on screens
     C*
     C           MSKDS     BEGSR
     C           DT        COMP 2                   47   No A2 disk prompt.
     C           DT        COMP 3                   48      A3  .
     C           DT        COMP 4                   49      A4...
     C                     ENDSR
     O*-----------------------------------
     OWORKSTN E                COMP
     O                                   K8 'COMPRESS'
     O                         SALLEA     4
     O                         SCD,1      5
     O                         SLH,1      9
     O                         SCD,2     10
     O                         SLH,2     14
     O                         SCD,3     15
     O                         SLH,3     19
     O                         SCD,4     20
     O                         SLH,4     24
     O                         CMDLIN    84
** CMD
Cmd4-Put on job queue  Cmd7-Cancel compress
Cmd2-Page back  Cmd4-Put on job queue  Cmd7-Cancel compress
```

Figure 18-10

Screen format member VTOCFRFM

```
*       1    . 2         3         4         5         6         7 .   . 8
     S*****************************************************************
     S* SCREEN NAME.. .    VTOCFRFM
     S* DESCRIPTION        VTOCFR screens
     S* PROGRAMMER         Chuck Lundgren (Iris Software, Inc.)
     S*   (c) COPYRIGHT 1990 Iris Software, Inc  - All Rights Reserved
     S* DATE WRITTEN..     January 1990
     S*
     S* VERSION DATE       FIX DESCRIPTION
     S* ---- ----- ------------------------
     S*****************************************************************
     SVTOCFR01            NYN                                   AEGHIL
     S*****************************************************************
     S* FORMAT NAME...   . VTOCFR01
```

```
S* PURPOSE        .  Free block display screen
S*********************************************************************
D           4 1 2Y               Y                CVTOC
D           1 1 8Y  Y          Y     Y            C1
D          17 113Y                                CFree disk blocks
DBAVTL     16 131Y
D          18 149Y                                CFree VTOC entries
DVAVTL     12 168Y
D          12 213Y                   40           CNo more than
DFREMAX     4 226Y                   40
D          44 231Y                   40           Cfree blocks can be dispX
Dlayed by this program
D          12 4 2Y                                CCommand keys
D          17 417Y               41               C1:One column/disk
D          11 436Y               42               C5:All disks
D           5 449Y                                C7:End
D          10 456Y               43               C8:Location
D           6 468Y               44  50           C9:Size
D          11 516Y                                C12:Compress
DSUBHED    11 7 2Y
D           4 722Y                   47           CA2 -
D           2 731Y                   47           CMb
DDMB,2      3 727Y                   47
D           4 742Y                   48           CA3 -
D           2 751Y                   48           CMb
DDMB,3      3 747Y                   48
D           4 762Y                   49           CA4 -
D           2 771Y                   49           CMb
DDMB,4      3 767Y                   49
D           8 8 2Y                       Y        CLocation
D           6 811Y                       Y        CBlocks
D           8 822Y                   57  Y        CLocation
D           6 831Y                   57  Y        CBlocks
D           8 842Y                   58  Y        CLocation
D           6 851Y                   58  Y        CBlocks
D           8 862Y                   58  Y        CLocation
D           6 871Y                   59  Y        CBlocks
DSCAD,1     6 9 3Y               Y
DSCAD,2     610 3Y               Y
DSCAD,3     611 3Y               Y
DSCAD,4     612 3Y               Y
DSCAD,5     613 3Y               Y
DSCAD,6     614 3Y               Y
DSCAD,7     615 3Y               Y
DSCAD,8     616 3Y               Y
DSCAD,9     617 3Y               Y
DSCAD,10    618 3Y               Y
DSCAD,11    619 3Y               Y
DSCAD,12    620 3Y               Y
DSCAD,13    621 3Y               Y
DSCAD,14    622 3Y               Y
DSCAD,15    623 3Y               Y
DSCAD,16    624 3Y               Y
DSCAD,17    6 923Y               Y
DSCAD,18    61023Y               Y
DSCAD,19    61123Y               Y
DSCAD,20    61223Y               Y
DSCAD,21    61323Y               Y
DSCAD,22    61423Y               Y
DSCAD,23    61523Y               Y
DSCAD,24    61623Y               Y
DSCAD,25    61723Y               Y
DSCAD,26    61823Y               Y
DSCAD,27    61923Y               Y
DSCAD,28    62023Y               Y
DSCAD,29    62123Y               Y
DSCAD,30    62223Y               Y
DSCAD,31    62323Y               Y
DSCAD,32    62423Y               Y
DSCAD,33    6 943Y               Y
DSCAD,34    61043Y               Y
DSCAD,35    61143Y               Y
DSCAD,36    61243Y               Y
DSCAD,37    61343Y               Y
DSCAD,38    61443Y               Y
DSCAD,39    61543Y               Y
```

DSCAD,40	61643Y	Y
DSCAD,41	61743Y	Y
DSCAD,42	61843Y	Y
DSCAD,43	61943Y	Y
DSCAD,44	62043Y	Y
DSCAD,45	62143Y	Y
DSCAD,46	62243Y	Y
DSCAD,47	62343Y	Y
DSCAD,48	62443Y	Y
DSCAD,49	6 963Y	Y
DSCAD,50	61063Y	Y
DSCAD,51	61163Y	Y
DSCAD,52	61263Y	Y
DSCAD,53	61363Y	Y
DSCAD,54	61463Y	Y
DSCAD,55	61563Y	Y
DSCAD,56	61663Y	Y
DSCAD,57	61763Y	Y
DSCAD,58	61863Y	Y
DSCAD,59	61963Y	Y
DSCAD,60	62063Y	Y
DSCAD,61	62163Y	Y
DSCAD,62	62263Y	Y
DSCAD,63	62363Y	Y
DSCAD,64	62463Y	Y
DSCSZ,1	6 911Y	Y
DSCSZ,2	61011Y	Y
DSCSZ,3	61111Y	Y
DSCSZ,4	61211Y	Y
DSCSZ,5	61311Y	Y
DSCSZ,6	61411Y	Y
DSCSZ,7	61511Y	Y
DSCSZ,8	61611Y	Y
DSCSZ,9	61711Y	Y
DSCSZ,10	61811Y	Y
DSCSZ,11	61911Y	Y
DSCSZ,12	62011Y	Y
DSCSZ,13	62111Y	Y
DSCSZ,14	62211Y	Y
DSCSZ,15	62311Y	Y
DSCSZ,16	62411Y	Y
DSCSZ,17	6 931Y	Y
DSCSZ,18	61031Y	Y
DSCSZ,19	61131Y	Y
DSCSZ,20	61231Y	Y
DSCSZ,21	61331Y	Y
DSCSZ,22	61431Y	Y
DSCSZ,23	61531Y	Y
DSCSZ,24	61631Y	Y
DSCSZ,25	61731Y	Y
DSCSZ,26	61831Y	Y
DSCSZ,27	61931Y	Y
DSCSZ,28	62031Y	Y
DSCSZ,29	62131Y	Y
DSCSZ,30	62231Y	Y
DSCSZ,31	62331Y	Y
DSCSZ,32	62431Y	Y
DSCSZ,33	6 951Y	Y
DSCSZ,34	61051Y	Y
DSCSZ,35	61151Y	Y
DSCSZ,36	61251Y	Y
DSCSZ,37	61351Y	Y
DSCSZ,38	61451Y	Y
DSCSZ,39	61551Y	Y
DSCSZ,40	61651Y	Y
DSCSZ,41	61751Y	Y
DSCSZ,42	61851Y	Y
DSCSZ,43	61951Y	Y
DSCSZ,44	62051Y	Y
DSCSZ,45	62151Y	Y
DSCSZ,46	62251Y	Y
DSCSZ,47	62351Y	Y
DSCSZ,48	62451Y	Y
DSCSZ,49	6 971Y	Y
DSCSZ,50	61071Y	Y
DSCSZ,51	61171Y	Y

```
DSCSZ.52    61271Y              Y
DSCSZ.53    61371Y              Y
DSCSZ.54    61471Y              Y
DSCSZ.55    61571Y              Y
DSCSZ.56    61671Y              Y
DSCSZ.57    61771Y              Y
DSCSZ.58    61871Y              Y
DSCSZ.59    61971Y              Y
DSCSZ.60    62071Y              Y
DSCSZ.61    62171Y              Y
DSCSZ.62    62271Y              Y
DSCSZ.63    62371Y              Y
DSCSZ.64    62471Y              Y
SCOMPRESS             NYN       Y                        BDG
S*********************************************************************
S* FORMAT NAME.       COMPRESS
S* PURPOSE            Compress request screen
S*********************************************************************
D           26 127Y             Y                CCompress All or Some DiX
Dsks
D            1 161Y  Y          Y    Y           C2
D           64 3 2Y                              CCOMPRESS ALL or compresX
Ds each disk individually          ALL,EACH
DSALLEA      4 367Y  YA         45     51  Y
D           64 5 2Y                    46        CCompress disk A1      X
D                                       Y,N
DSCD,1       1 567Y  YA    56   46   4652  Y
D           64 6 2Y                    46        CLocation for A1 free spX
Dace                               LOW,HIGH
DSLH,1       4 667Y  YA         46   4652  Y
D           64 8 2Y                    47        CCompress disk A2      X
D                                       Y,N
DSCD,2       1 867Y  YA    53   47   4753  Y
D           64 9 2Y                    47        CLocation for A2 free spX
Dace                               LOW,HIGH
DSLH,2       4 967Y  YA         47   4753  Y
D           6411 2Y                    48        CCompress disk A3      X
D                                       Y,N
DSCD,3       11167Y  YA    54   48   4854  Y
D           6412 2Y                    48        CLocation for A3 free spX
Dace                               LOW,HIGH
DSLH,3       41267Y  YA         48   4854  Y
D           6414 2Y                    49        CCompress disk A4      X
D                                       Y,N
DSCD,4       11467Y  YA    55   49   4955  Y
D           6415 2Y                    49        CLocation for A4 free spX
Dace                               LOW,HIGH
DSLH,4       41567Y  YA         49   4955  Y
D           6024 2Y
```

Figure 18-11

Procedure VTOCFR

```
*
* Procedure   VTOCFR
* Parameters
*   1 FREE     Display free blocks screen
*     COMPRESS Display compress request screen
*   2 ONE      Show free blocks for each disk in a separate column
*     ALL      Show free blocks for all disks in all columns
*   3 LOCATION Sort free blocks by location
*     SIZE     Sort free blocks by size (not avail  for "ONE" parameter)
*
// LOCAL OFFSET-171,BLANK-41
// IF  ?1?-        EVALUATE P1-'FREE'
// IF  ?2?-        EVALUATE P2-'ALL'
// IF  ?3?-        EVALUATE P3-'LOCATION'
// IFF ?1?-'FREE'  IFF ?1?-'COMPRESS'      GOTO BOMB
// IFF ?2?-'ONE'   IFF ?2?-'ALL'           GOTO BOMB
// IFF ?3?-'SIZE'  IFF ?3?-'LOCATION'      GOTO BOMB
// IF  ?2?-'ONE'   IFF ?3?-'LOCATION'      GOTO BOMB
// LOCAL OFFSET-171,DATA-'?1?'
// LOCAL OFFSET-179,DATA-'?2?'
// LOCAL OFFSET-182,DATA-'?3?'
// LOCAL OFFSET-190,DATA-'EACH'            Default for compress
// LOCAL OFFSET-194,DATA-'YHIGH'           Default for A1 COMPRESS
```

```
// LOCAL OFFSET-199,DATA-'YLOW '                               A2 COMPRESS
// LOCAL OFFSET-204,DATA-'YHIGH'                               A3 COMPRESS
// LOCAL OFFSET-209,DATA-'YLOW '                               A4 COMPRESS
*
// LOAD VTOCFR
// RUN
*
// IFF  ?L'171,8'?='COMPRESS'             RETURN
*
// LOAD VTOCCM
// RUN
*
// IF ?L'215,1'?='E'                      RETURN
// IF ?L'190,4'?='ALL ' IF ?L'215,1'?='J' JOBQ ,COMPRESS,ALL
// IF ?L'190,4'?='ALL ' IF ?L'215,1'?='C' COMPRESS ALL
// IF ?L'190,4'?='EACH' IF ?L'215,1'?='J' JOBQ ,VTOCCM
// IF ?L'190,4'?='EACH' IF ?L'215,1'?='C' VTOCCM
// RETURN
*
// TAG BOMB
// PAUSE 'Illegal parameter in VTOCFR'
```

Figure 18-12

Procedure VTOCCM

```
*
*  Procedure   VTOCCM
*
// IF ?L'194,1'?='Y'                     COMPRESS A1,FREE?L'195,4'?
// IF ?L'199,1'?='Y' IF ?L'214,1'?>'1'   COMPRESS A2,FREE?L'200,4'?
// IF ?L'204,1'?='Y' IF ?L'214,1'?>'2'   COMPRESS A3,FREE?L'205,4'?
// IF ?L'209,1'?='Y' IF ?L'214,1'?>'3'   COMPRESS A4,FREE?L'210,4'?
```

Figure 18-13

COMPRESS screen

```
                          Compress All or Some Disks

COMPRESS ALL or compress each disk individually        ALL,EACH EACH

Compress disk A1                                           Y,N Y
Location for A1 free space                             LOW,HIGH HIGH

Compress disk A2                                           Y,N Y
Location for A2 free space                             LOW,HIGH LOW











Cmd4-Put on job queue  Cmd7-Cancel compress
```

Re-creating Subroutine SUBRF5

If you don't have assembler subroutine SUBRF5, you can re-create it with procedure MKSUBRF5 (you don't need IBM's Assembler Language Program Product to install SUBRF5). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRF5. You need to run MKSUBRF5 only once because SUBRF5 is subsequently linked into program VTOCFM when it is compiled.

```
// * 'Re-creating R-module SUBRF5 in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-301,DATA-'00000187'     Number of $MAINT records
// LOCAL OFFSET-309,DATA-
'D9E2E4C2D9C6F540400000CA000000000000006C4000000009900006730000000438B'
// LOCAL OFFSET-373,DATA-
'1204150531000000801300000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRF5 member to insert object code
// LOAD $FEFIX
// RUN
HDR   3858 SUBRF00000
PTF   2059 RSUBRF5 99,,#RPGLIB
DATA 4504 00 0000 E2D8E2E4C2D9C6F5000000000C0000000000000000000000000000000000000000
DATA CF78 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 3F28 00 0040 E3330C33F2B7C4E3E4C2D9C6F540F140F340404040403401C18034020C1A13408
DATA 271E 00 0060 01A5F4000A9CF0CC7536A20C682C020C7C00C087C1A83801003120282410161S
DATA C0UB 00 0080 E330006401A6750105AC05000C8435A20C882C020C1C003C030C88C08F01023B
DATA 1388 00 00A0 0101A675010B4C06000CBA36A20C8E3C020C7C003C0072E3A2610'A1E110000801
DATA F8B8 00 00C0 E33000950308C88C087010238010 1A675010B4C05000C8A38A20C472C020C7C00
DATA AE5F 00 00E0 3C030C88C087010238010 1A675010E4C05000C8A36002F27231F1A1E120A0602
DATA 5F8E 00 0100 E3330C8A20C472C030C7C003C030C88C087010236010145750111 4C05000C8A
DATA A4C8 00 0120 C2A1021E7807 0F38A20C826C071500F401140758A20F0F020026E1813 0F080802
DATA D9CE 00 0140 E331 0CFA0C7C0C7C2C010C7C37C087 01A635010146750114 4C03000C84C2A102
DATA F04E 00 0160 1E3584 20C858C021600F40140758A20F3C020C7C07C087002E201C181 00C070301
DATA 83F8 00 0180 E33201 2001A6380101A67501174C03000C8435010 1A57501024001070C50F781
DATA AEB1 00 01A0 073CF10C75F2E7681C010C77037601 01E2020B0CC10C7800322720191100060
DATA E1C3 00 01C0 E330C16E0C6480C2020C4AF781403 C020C7C02C087 01A6370 10C4E4C05000CB3
DATA C8B4 00 01E0 2C020C7U0527F020C 7C07C08 F01A63801 0L484C05000 07028241F1818120000801
DATA 0835 00 0200 E3320 1910C830 2010C3401 0C800001 0C800C77F2040730F 20C75F2B70CE2020
DATA 8FC6 00 0220 0F010C790C4CF3C154C2A1000035010 1A67501 1A4C0000002C2 11F150B0C0803
DATA 8780 00 0240 E37F01C1DC780E010 1A60C82C3A10000C2A20000C08700030 40801 010407 0CB4
DATA 702D 00 0260 0C5C3C1 80C8E080 70C840C840E020C7C0C7CF2 30 0020 2B2725211 01817070601
DATA 8F42 00 0280 E33201 F4043A010C843F010C8EF1011ACU8 70000340 80021034A10C7134A20C74
DATA C751 00 02A0 C2A10C85C7A20C7A4C010 10C6C88020000880301 00 70F 900 2823 1F1817 130804
DATA 82C8 00 02C0 E33802 2B00F2040 37F3900 70F901F2040 37F780 10201 07E2020 13F010C88F101
DATA 7AC6 00 02E0 2C35A10C7 136A20C74C087000C9C6000000000040 0000A10CA080000002420 19
DATA 0F8B 00 0300 E311023002450000090000000000000000000000000000000000000000000000
DATA 863B 00 0320 0000000000000000000000000000000000000000000000000000000000000001
DATA 8051 00 0340 E3790C6E000003FFFFFFF00C10000800C0018014 8F0F0F0F0F0F0F0F0000084A00
DATA 38F8 00 0360 0A1400CA19000A74000A7A000A7000000000000000000000000000000000000000
DATA DEGA 00 0380 E33A0CC9E2F4C209C6F540604 0C3986174859 8887 88A340408 35040F1F9F6F86B
DATA 8834 00 03A0 40C7814 9484 0E 3484 0029881A3A8B59 94 04 0C1 9 3 93 4 0 9 8 8 9 4 7 8 8A3A34 0 9 8 8 6 0 0
DATA 747F 00 03C0 E30B0CCFA38 59 8A8 8 5840000000000000000000000000000000000000000
DATA 944F 00 03E0 0000000000000000000000000000000000000000000000000000000000000000
DATA C3C8 00 0400 C50000CF00000000000000000000000000000000000000000000000000000000
DATA 4E83 00 0420 0000000000000000000000000000000000000000000000000000000000000000
DATA B111 00 0440 815C0000000000000000000000000000000000000000000000000000000000000
DATA 50E6 00 0460 0000000000000000000000000000000000000000000000000000000000000000
DATA 368E 00 0480 4040404040404040404040404040404040404040404040404040404040404040
```

Continued

```
DATA 5594 00 04A0 4040404040404040404040404040404040404040404040404040404040404040
DATA B368 00 04C0 4040404040404040404040404040404040404040404040404040404040404040
DATA C338 00 04E0 4040404040404040404040404040404040404040404040404040404040404040
END  7592
```

Differences Between Actual Disk Space and CATALOG Listing

answered by NEWS 3X/400 Staff

Q After adding up all my program products, user libraries, and data files, I found 2,500 blocks unaccounted for. Does the system eat up this much space for diagnostic purposes?

A The system uses about 650 blocks (depending on the size of your system) that do not show up in the VTOC. All scratch or work files also do not appear in the VTOC listing. We recommend you run a compress and then run a CATALOG by location to get an accurate block count. And instead of running a CATALOG, you could run subroutine VTOCFR (*Displaying Free Disk Space*, page 625), which lets you see how much free space you have.

Retrieving a File's or Library's Users

by Perry Gardai

program by Matthew Henry

Code on diskette:
Procedure TESTU
RPG program TESTU
Screen format member TESTUFM

Often in the day-to-day operation of a S/36 data processing shop, an operator may need to know whether a particular library or file is in use. This is especially true when dedicated procedures such as RENAME, DELETE, or CONDENSE need to be performed. While IBM provides some identification data via various console operations, the data is not library- or file-specific and is often awkward to access, making it difficult to determine which users are tying up the library or file in question.

Two utilities appearing elsewhere in this book, TESTUL and TESTUF, offer procedurally driven methods to discover the library or file users. This utility, TESTU, offers an interactive, well-formatted, user-friendly utility that displays the same library and file usage information. Before we examine how on-line program TESTU is constructed, let's first review its predecessors to see what makes TESTU another useful implementation of the basic techniques put forth in the previous articles.

The information requested by TESTUL and TESTUF is presented to the user via OCL message statements. These statements, embedded within a loop, call a program that accesses an assembler subroutine (SUBRUL or SUBRUF) via the EXIT operation and then loads the information into the LDA. The LDA data is substituted into the OCL message statement, and the message is displayed on the screen while statements within the loop test a counter for EOJ. The cycle is repeated as often as there are jobs or workstations using the target library or file. While such procedurally driven implementations are useful for batch applications, they have some inherent constraints.

Functions such as creating column headings on the screen, rolling back and forth through the data, and changing from one target library or file to another prove difficult if not impossible. A second drawback is simply execution speed. The time required to initiate the program that accesses the modified subroutine and to translate and execute all of the OCL statements in the loop is far more than that required by an on-line program.

Unlike the unformatted data the OCL implementation presents, TESTU provides column headings that organize the various data elements retrieved by the subroutines. Program TESTU offers other advantages too. The Roll key function allows the operator to review entries that might have rolled off the screen in the OCL versions. Also, Command key 1 takes the operator back to the first screen to select another library or file to review. And once program TESTU is loaded and running, the screen response is almost instantaneous, a vast improvement over the comparatively slow screen messages issued by the procedural versions.

Using the TESTU Utility

To create program TESTU, the two assembler subroutines — SUBRUL and SUBRUF — must be stored in #RPGLIB. (See *Retrieving a Library's Users* (page 272) and *Retrieving a File's Users* (page 205) for how to create the SUBRUL and SUBRUF subroutines.) Once the subroutines are created and stored, any RPG program can access them, and the information returned by the subroutines can be used in the same way as data from a file; these subroutines are the fundamental building blocks of program TESTU.

To run the TESTU procedure (Figure 18-14), simply key in TESTU. With the exception of the section of code that accesses the external subroutines, the program that TESTU calls is a straightforward two-screen program that uses Roll key and Command key logic (see Figure 18-15 for the screen format member specifications). The NAME screen (Figure 18-16) prompts the operator to supply the name of the target library or file and to designate with the letter L or F which it is. The status screen (Figure 18-17) contains the library or file usage data along with appropriate headings, column titles, and command key instructions.

Now let's look at the program itself (Figure 18-18). The initialization section accepts the library or file name and designation (L or F) from the LDA on the first cycle, thus enabling the NAME screen to be bypassed. The program performs some minor edits and then drops into a DOUEQ loop.

Within the loop, the designation F or L is established, and the appropriate external subroutine is accessed via the EXIT command. The three RLABL statements provide the data for the library or file name, the index *x* (a counter), and the data structure named JOBDS. Once the program knows the name of the library or file in question, the index *x* counts each job using that library or file. The JOBDS data structure is subdivided into various data fields that are subsequently moved to the corresponding screen data fields. Next, the screen data fields are redefined as the data field SCREEN in a data structure, and SCREEN is moved to the current element of the OT array in preparation for output. This loop is repeated either until there are no more users and user information for the assembler subroutine to retrieve or a maximum of 20 times (the maximum number of lines reserved on the STATUS screen). If there are more than 20 entries, the program performs a Roll key procedure to scroll from page to page.

Before the STATUS screen is output, the OT array is sorted in descending sequence via the SORTA command. The sort causes any blank entries read by the external subroutine to be sorted at the end of the list, thus eliminating blank lines on the screen when it is displayed. At the end of the calculations, subroutine SUBINF controls the Roll key functions enabled on the STATUS screen. Finally, Command key 1 returns the operator to the NAME screen to select another library or file for inspection. Command key 7 ends the program.

Remember, the next time a user ties up a library or file you need for a dedicated system function, the TESTU utility provides a fast, easy, and efficient method for identifying the culprit.

Figure 18-14

Procedure TESTU

```
// LOCAL OFFSET-1,DATA-'?1?',BLANK-9
// IF ?2?/ IF LOAD-'#PTFLOG,?1?' EVALUATE P2-L
// LOCAL OFFSET-9,DATA-'?2?'
// LOAD TESTU
// RUN
```

Figure 18-15

SFGR specifications TESTUFM for NAME and STATUS screens

```
*        1         2         3         4         5         6         7         8
     SNAME      00        NN        Y                          12345
     DFL0001    37 3 2Y               Y                C     .                 X
     D      .
     DFA0001     1 4 2Y               Y                C
     DFA0001     1 4 4Y  Y          Y     Y            CP
     DFA0002    31 4 6Y                                C                       X
     D
     DFA0002     1 438Y               Y                C
     DFL0002     1 5 2Y               Y                C
     DFL0003    24 5 4Y                                CEnter file/library nameX
     D.
     DFL0004     8 529Y  YB                     Y N
     DFL0005     1 538Y               Y                C
     DFA0003     1 6 2Y               Y                C
     DFA0002    33 6 4Y                                C                       X
     D
     DFA0004     1 638Y               Y                C
     DFA0005     1 7 2Y               Y                C
     DFA0001    33 7 4Y                                CEnter L for library or X
     D
     DFA0006     1 738Y               Y                C
     DFA0007     1 8 2Y               Y                C
     DFA0002    24 8 4Y                                C      F for file       X
```

```
D
DFA0003     1 829Y  YA                    Y N
DFA0004     6 831Y                               C
DFA0008     1 838Y            Y                  C·
DFA0009     1 9 2Y            Y                  C:
DFA0011    33 9 4Y                               C                   X
D
DFA0010     1 938Y            Y                  C:
DFA0002     110 2Y            Y                  C:
DFA0001    3310 4Y            Y
DFA0003     11038Y            Y                  C:
DFL0006    3711 2Y            Y                  C:. ..PRESS ENTER TO ACCX
DEPT NAME... .:
SSTATUS              NY                                  AG15
DFA0001     9 121Y                               CUsers for
DFA0002     4 131Y
DFA0002     8 136Y
DFA0003    76 2 1Y                    Y          CJob      Proc      ProgrX
Dam  User     Initial  Start time  Share
DFA0003  1600 3 1Y
DFL0023    7023 2Y            Y
DFL0024    7924 2Y                               CCmd1-New file/lbrary   X
DCmd7-End program   Roll-Page   Enter-Update
```

Figure 18-16

Screen prompt for library or file name

```
..  ............. ........... ... ..
:                                  :
: Enter file/library name: EDITPROF :
:                                  .
: Enter L for library or           .
:       F for file . . . . F       ·
:                                  :
: **Enter a name                   :
:....PRESS ENTER TO ACCEPT NAME.. ..:
```

Figure 18-17

Screen displaying user information

```
                   Users for file EDITPROF
Job       Proc     Program  User     Initial  Start time  Share
W9101354  FSEDIT   FSED2    RALPH    FSEDIT   10:23am     SHRMM
W8101354  FSEDIT   FSED2    GARRISON SSRNEW   10:10am     SHRMM
W7101354  FSEDIT   FSED2    MONTE    FSEDIT   12:07pm     SHRMM
W6101354  FSEDIT   FSED2    OBERG    PROBLOG  11:15am     SHRMM
W5101354  FILE#    FILE     PENNY    FILE     12:12pm     SHRRM
W4101354  FSEDIT   FSED2    MERLE    FSEDIT   09:35am     SHRMM
W3101354  FSEDPF   FSEDPF   GARY     LIBR     10:03am     SHRRM
W2101354  FSEDIT   FSED2    REBECCA  FSEDIT   09:21am     SHRMM
W1101354  FSEDIT   FSED2    MEL      FSEDIT   10:13am     SHRMM
WB101354  FSSTAT   FSSTAT   ZIMMER   FSDIAG   08:01am     SHRRM
WA101354  FSEDIT   FSED2    INGRID   SSRUPD   09:18am     SHRMM


 **End of list
 Cmd1-New file/library   Cmd7-End program   Roll-Page   Enter-Update
```

Figure 18-18

Program TESTU

```
*.. .  1 .. .. 2 . .  . 3 . . ... 4 ... ... 5 ... .  6 . .  . 7 ... . 8
0001 H     064                     B       1                       TESTU
0002 F**************************************************************
0003 F* Program: TESTU   Written by: Matthew P. Henry              *
0004 F*                             Thanks to Mel Beckman          *
```

```
F* This program prompts for a library or file and displays a list *
F* of all programs and users using that library or file           *
F* Comments: The report on the screen is sorted in descending      *
F*           order  This is done for simplicity; otherwise, all    *
F*           the blank entries would be first  You can take the    *
F*           SORTA operation out if it is confusing  I like it     *
F*           because I can follow through a list of all the        *
F*           terminals.                                            *
F******************************************************************
F*                                                                 *
F* Flags:  (1-ON, 0-OFF)                                           *
F* FIRST  - ON  initialization completed                           *
F* RPT    - ON  repeat of list cycle                               *
F*          OFF prompt for new name                                *
F* LSTEND - ON  end of list reached                                *
F* NONE   - ON  no programs or users for specified name            *
F*                                                                 *
F* Command keys.                                                   *
F* KA - CMD1 Request to enter new name/type                        *
F* KG - CMD7 Request to end program                                *
F*                                                                 *
F******************************************************************
FWORKSTN CP   F     2000            WORKSTN
F                                              KINFDS EXCPOS
F                                              KINFSR SUBINF
E                    SHR    10  10  5
E                    MSG     1   4 70
E                    OT         20 80   D
E                    AJT         4  1
E                    SJT         7  1
IWORKSTN NS       1 CP
I                                        2   9 FILNAM
I                                       10  10 TYPE
I        NS
IJOBDS       DS
I                                        1   8 USERID
I                                        9  16 JOBNAM
I                                        9  10 JWS
I                                       11  160JT
I                                       17  24 FSTPRC
I                                       25  32 CURPRC
I                                       33  40 PRGNAM
I                                       41  460JSTIME
I                                        1  46 JINFO
I                                       47  470SHRLVL
IEXCPDS      DS
I                                     *STATUS  STATUS
I                                       23  26ORCODE
I            DS
I                                        1  80 SCREEN
I                                        1   8 SJOBNA
I                                       10  17 SCURPR
I                                       19  26 SPRGNA
I                                       28  35 SUSERI
I                                       37  44 SFSTPR
I                                       46  52 SJT
I                                       54  55 JOFLAG
I                                       58  62 SHRTXT
I            DS
I                                        1   2 NULL
I                                        1   1 NULL1
I                                        2   2 NULL2
I           UDS
I                                        1   8 LDAFIL
I                                        9   9 LDATYP
C*
C* Initialization
C*
C           FIRST     IFEQ ' '
C                     BITOF'01234567'NULL1
C                     BITOF'01234567'NULL2
C           LDAFIL    IFEQ *BLANKS
C                     MOVE 1         RPT     10
C                     ELSE
C                     MOVE LDAFIL    FILNAM
C                     MOVE LDATYP    TYPE
```

```
C                     Z-ADDO         X       30
C                     END
C                     MOVE 1         FIRST   1
C                     END
C*
C* Main section
C*
C                     MOVE *BLANKS   ERROR  70
C   KA                MOVE 0         RPT     10
C  NKA                MOVE 1         RPT
C           FILNAM    IFEQ *BLANKS
C                     MOVEAMSG,3     ERROR
C                     MOVE 0         RPT
C                     END
C           RPT       IFEQ 0
C                     Z-ADDO         X       30
C                     END
C           TYPE      IFEQ *BLANKS
C                     MOVE 'F'       TYPE
C                     END
C  NKG      RPT       IFEQ 1
C                     Z-ADD1         Y       30
C                     MOVE X         HOLD    30
C                     MOVEA*BLANKS   OT
C           JOBNAM    DOUEQ*BLANKS
C                     Z-ADDO         NONE    10
C           TYPE      IFEQ 'F'
C                     MOVE 'file'    STYPE   4
C                     EXIT SUBRUF
C                     RLABL          FILNAM
C                     RLABL          X
C                     RLABL          JOBDS
C           SHRLVL    ADD  1         SL      20
C                     MOVE SHR,SL    SHRTXT  5
C                     ELSE
C                     MOVE 'libr'    STYPE   4
C                     EXIT SUBRUL
C                     RLABL          FILNAM
C                     RLABL          X
C                     RLABL          JOBDS
C                     END
C           JOBNAM    IFNE *BLANKS
C           USERID    IFEQ *BLANKS
C                     MOVEL'MRT JOB' SUSERI
C                     ELSE
C           JWS       IFEQ NULL
C                     MOVE '??'      JWS
C                     MOVE 'HELP KEY'CURPRC
C                     END
C                     MOVE USERID    SUSERI
C                     END
C                     MOVE JOBNAM    SJOBNA
C                     MOVE FSTPRC    SFSTPR
C                     MOVE CURPRC    SCURPR
C                     MOVE PRGNAM    SPRGNA
C           JSTIME    IFNE *ZEROS
C                     MOVE JSTIME    JT
C                     MOVE 'JQ'      JQFLAG
C                     ELSE
C                     MOVE '  '      JQFLAG
C                     END
C           JT        IFGT 115959
C                     MOVEA'pm'      SJT,6
C                     ELSE
C                     MOVEA'am'      SJT,6
C                     END
C           JT        IFGE 130000
C                     SUB  120000    JT
C                     ELSE
C           JT        IFLT 010000
C                     ADD  120000    JT
C                     END
C                     END
C                     MOVEAJT        AJT
C                     MOVEAAJT,1     SJT,1
C                     MOVEAAJT,2     SJT,2
```

```
C                     MOVEA' '       SJT,3
C                     MOVEAAJT,3     SJT,4
C                     MOVEAAJT,4     SJT,5
C                     MOVEASCREEN    OT,Y
C                     MOVE 0         LSTEND  10
C                     ELSE
C*-> TEST FOR END OF LIST
C           X         IFEQ 0
C                     MOVE 1         NONE    10
C                     MOVE 1         LSTEND  10
C           TYPE      IFEQ 'F'
C                     MOVEAMSG,1     ERROR
C                     ELSE
C                     MOVEAMSG,2     ERROR
C                     END
C                     ELSE
C                     MOVE 1         LSTEND  10
C                     MOVEAMSG,4     ERROR
C                     END
C                     END
C*-> INCREMENT POINTERS
C                     ADD  1         X
C                     ADD  1         Y
C           Y         IFGT 20
C                     MOVE *BLANKS   JOBNAM
C                     END
C                     END
C*-> END OF DOUNTIL
C                     MOVE HOLD      X
C           NONE      IFEQ 0
C                     SORTAOT
C                     END
C                     EXCPTLIST
C                     ELSE
C                     EXCPTPROMPT
C                     END
C*-> END OF IF
C   KG                SETON                     LR
C*
C* Subroutine to control roll-up and roll-down
C*
CSR         SUBINF    BEGSR
C                     SETOF                     KA
C                     MOVE 1         RPT
C           STATUS    IFEQ 01122
C           LSTEND    IFEQ 0
C                     ADD  20        X
C                     END
C                     ELSE
C           STATUS    IFEQ 01123
C                     MOVE 0         LSTEND
C                     SUB  20        X
C                     END
C                     END
C           X         IFLT 0
C                     Z-ADD0         X
C                     END
CSR                   ENDSR'*DETC'
OWORKSTN E                LIST
O                                   K8 'STATUS   '
O                         STYPE      4
O                         FILNAM    12
O                         OT      1612
O                         ERROR   1682
OWORKSTN E                PROMPT
O                                   K8 'NAME     '
O                         FILNAM     8
O                         TYPE       9
O                         ERROR     79
** Share levels
SHRRMSHRRRSHRMRNOSHR                         SHRMM
** Messages
**File currently not in use, Cmd1 to enter another name
**Library currently not in use, Cmd1 to enter another name
**Enter a name
**End of list
```

Explanation of the Job Queue

answered by Mike Patton

Q Our company recently upgraded from a S/36 5362 to a full-sized 5360 on which the SSP was loaded before we took delivery. Since the upgrade, every job we place on the job queue is released immediately to the user area for execution — something that did not happen on the 5362. Is there a command or procedure we can use to make the job queue work properly?

A It is possible that your jobs are being released immediately from the job queue because the maximum number of active job-queue jobs (Max Active Jobs on the job queue status display, Figure 18-19) is set to a larger number. Alternatively, you may be using an excessively high value for the maximum number of active job-queue jobs at a particular priority level (Max for PRTY on the job queue status display). To find out whether this is the case, enter D J at the system console to display the job queue status.

Notice the settings for Max Active Jobs and Max for PRTY in Figure 18-19. In this example, a maximum of five jobs from the job queue may be active at one time. Of these five active jobs, a maximum of one job each may be active with priority levels 5, 4, 3, 2, or 1. One job with priority level 0 also is allowed, but jobs placed on the job queue with priority 0 are not executed automatically; they must be released for execution by the operator with the "S J,jobname" command. For this reason, they are not counted in the maximum number of active job-queue jobs.

To change these values, take option 10 on the JOBQUEUE menu of the job queue status display. You may then select the JOBQ option to limit the total number of active jobs, or you may select an individual job priority

Figure 18-19
Job queue status display

```
     Complete                  JOB QUEUE STATUS                              W5
     Jobs in Queue:   0 of    100        JOBQ PRTY  STOPPED    0
        Max Active Jobs. 5    Max for PRTY 5 1  4 1  3:1  2:1  1 1  0.1
                                                                PRIORITY
 POS  JOBNAME   PROC/DOC    LIBR/FLDR    USER       STATUS      JOBQ PROC

SYS-5689 The job queue is empty now



 Cmd7-End      Cmd8-Help     Cmd15-Update      Cmd16-Restart       Roll-Page
--------------------------------------
                              JOBQUEUE
                         Jobs on the job queue
   1  Display specific job(s)
   2  Put a job on the queue
   3  Cancel a job
   4  Hold a job
   5. Release a job
Ready for option number or command

                                                          COPR IBM Corp  1986
```

level for which you want to adjust the maximum number of active jobs. Valid maximums range from 0 to 50. Once you set a maximum value, it remains effective until you change it with the G J (Change Job Queue) command or with JOBQUEUE menu option 10 — or until the job queue is rebuilt by the system.

Manipulating the Job Queue

by Lisa A. Hendricks

Although moving a job from one priority to another normally is a five-step process, you can accomplish the same operation with much less effort. First, stop the JOBQ. Then place one job on the JOBQ for each priority from 1 through 5, hold all five of the jobs, and restart the JOBQ. The five "dummy" jobs sit in the JOBQ until you remove them. Finally, when a user places a job in the JOBQ, simply move his or her job to the position after the held "dummy" job that has the appropriate priority.

You also can speed DisplayWrite printing time by assigning a maximum of four jobs to priority 2 in the JOBQ. By reserving priority 2 for DisplayWrite, you eliminate the conflicts that arise when RPG programs and DisplayWrite jobs run concurrently from priority 2.

Executing an OCL Statement on the Job Queue

by Mel Beckman

Code on diskette:
Procedures J//, JOCL
RPG program JOCL

Many times I'd like to put an OCL statement on the job queue. For example, I might want to send myself a message, via the // MSG statement, when the jobs currently on the queue finish. Or I might want to hold up the queue until a certain time using the // WAIT statement. Or when I put a large stack of jobs on the queue at night, I might like to execute a // POWER OFF statement as the last step.

Unfortunately, the JOBQ command lets you place only procedures on the job queue — not OCL statements. You can overcome this restriction, however, by using the RPG program and pair of procedures shown in Figures 18-20, 18-21, and 18-22. Placing the two procedures and the compiled RPG program in #LIBRARY lets you put any OCL statement on the job queue simply by preceding the // statement with a J. For example, to put a // MSG statement on the job queue, key:

```
J// MSG MEL,JOB HAS FINISHED
```

This statement invokes the J// procedure (Figure 18-20), passing the OCL statement to be queued on the procedure parameter line. Procedure J// must have the program data attribute set (you set this attribute from the end-of-job screen in source editors such as FSEDIT, DSU, and SEU). Procedure J// runs RPG program JOCL (Figure 18-21), which reads a workstation file to retrieve the procedure command line, copy it, and put it in the LDA. Because of the program data attribute of procedure J//, the first workstation read operation performed by program JOCL gets the procedure command line as data, which it then stores in LDA positions 393 through 512 (to avoid conflicts with utilities such as POP).

Next, procedure J// puts procedure JOCL (Figure 18-22) on the job queue. When procedure JOCL runs, it "inherits" the LDA (which contains the OCL statement image) from procedure J//. Procedure JOCL substitutes LDA positions 393 through 512 into a statement starting with //, and the system interprets the resulting statement as an OCL statement.

Although I use J// for only MSG, WAIT, and POWER OCL statements, you also could use it for CANCEL, CHANGE, EVOKE, START, STOP, and VARY statements.

Figure 18-20
Procedure J//

```
// LOAD JOCL
// RUN
// JOBQ ,JOCL
```

Figure 18-21
Program JOCL

```
*         1         2         3         4 .   . . 5  .      6       . 7 . .   . 8
0001 H                                                                   JOCL
0002 F*
0003 F* Copy the procedure command line into the LDA
0004 F*
0005 FWORKSTN CP  F      120            WORKSTN
0006 F                                               KFMTS  *NONE
0007 F                                               KID    WSID
0008 IWORKSTN
0009 I                                       1 120 OCL
0010 I            UDS
0011 I                                     393 512 LDA
0012 C                      SETON                      LR
0013 C                      MOVE OCL       LDA
0014 OWORKSTN D         LRNLR
0015 O                                  K5 'DUMMY'
```

Figure 18-22
Procedure JOCL

```
// ?L'393,120'?
```

Changing Procedures Already Enqueued on the Job Queue

answered by Matthew Henry

Q Suppose I bring up a procedure member, make a change, save it, and then submit it to be executed on the job queue. If after submitting it to be executed, I go back into the member, make another change, and save it before the first procedure starts to be executed, which version will actually be executed? From my testing, it appears the second version (or the one most recently updated) would be executed.

On a mainframe, a job submitted to be executed will “carry along” a copy of the JCL. If I subsequently change the JCL, or cancel the edit, the submitted version is executed exactly as I submitted it. The S/36 appears to submit only the library member name, and whatever is in the member at that time is what will be executed. Why? Exactly what does the S/36 do when it executes a job from the job queue?

A A job run from the job queue is started just like a job run from a terminal. The only things “carried” with the job are a copy of the terminal’s local data area, switches, and session configuration information because making a copy of the entire OCL for a batch job would require additional storage on the S/36. The operating system reads the OCL procedure when the time comes to execute it. Thus, a procedure placed on the job queue but which has not started executing could be held and modified (just by changing the procedure within the library) before it is run.

Displaying and Updating of the LDA and UPSI Switches

by John E. King, III

Code on diskette:
Procedure LDA
Screen format member LDAFM

BitStop has featured several S/36 LDA display procedures in the past, but procedure LDA (Figure 18-23) goes a little further than the others by allowing you to use Command key 1 to toggle between the system LDA and the user LDA.

Procedure LDA uses the S/36 EVALUATE statements to retrieve the LDA data 100 bytes at a time and assign the data to parameters 3 through 8. The procedure then sets parameters 9 through 16 to 0 or 1, depending on the UPSI switch settings. The LDA data and UPSI switch settings are displayed via the parameters on a prompt screen (Figure 18-24 shows the prompt screen, and Figure 18-25 shows the S- and D-specs). Command key 2 serves as a toggle to show you two screens, one with the system LDA and the UPSI switch settings, the other with the user LDA and the UPSI switch settings. The facility also lets you update the LDA data or the UPSI switch settings.

Figure 18-23
Procedure LDA

```
* PROGRAM DISPLAYS THE LOCAL DATA AREA
// EVALUATE P1-?WS?  P2-'USER  '
// TAG TOP
// LOCAL AREA-?2?
// EVALUATE P3-'?L'1,100'?' P4-'?L'101,100'?' P5-'?L'201,100'?' P6-'?L'301,100'?'
// EVALUATE P7-'?L'401,100'?' P8-'?L'501,12'?'
```

```
// IF SWITCH1-0 EVALUATE P9-0
//       ELSE   EVALUATE P9-1
// IF SWITCH2-0 EVALUATE P10-0
//       ELSE   EVALUATE P10-1
// IF SWITCH3-0 EVALUATE P11-0
//       ELSE   EVALUATE P11-1
// IF SWITCH4-0 EVALUATE P12-0
//       ELSE   EVALUATE P12-1
// IF SWITCH5-0 EVALUATE P13-0
//       ELSE   EVALUATE P13-1
// IF SWITCH6-0 EVALUATE P14-0
//       ELSE   EVALUATE P14-1
// IF SWITCH7-0 EVALUATE P15-0
//       ELSE   EVALUATE P15-1
// IF SWITCH8-0 EVALUATE P16-0
//       ELSE   EVALUATE P16-1
// PROMPT MEMBER-LDAFM,FORMAT-SCREEN01,LENGTH-'2,6,100,100,100,100,100,12,+
1,1,1,1,1,1,1,1'
// IF ?CD?/2007 RETURN
// LOCAL OFFSET-1,AREA-?2?,DATA-'?3?',BLANK-*ALL
// LOCAL OFFSET-101,AREA-?2?,DATA-'?4?'
// LOCAL OFFSET-201,AREA-?2?,DATA-'?5?'
// LOCAL OFFSET-301,AREA-?2?,DATA-'?6?'
// LOCAL OFFSET-401,AREA-?2?,DATA-'?7?'
// LOCAL OFFSET-501,AREA-?2?,OATA-'?8?'
// IF ?9?/0  SWITCH 0XXXXXXX
//       ELSE SWITCH 1XXXXXXX
// IF ?10?/0 SWITCH X0XXXXXX
//       ELSE SWITCH X1XXXXXX
// IF ?11?/0 SWITCH XX0XXXXX
//       ELSE SWITCH XX1XXXXX
// IF ?12?/0 SWITCH XXX0XXXX
//       ELSE SWITCH XXX1XXXX
// IF ?13?/0 SWITCH XXXX0XXX
//       ELSE SWITCH XXXX1XXX
// IF ?14?/0 SWITCH XXXXX0XX
//       ELSE SWITCH XXXXX1XX
// IF ?15?/0 SWITCH XXXXXX0X
//       ELSE SWITCH XXXXXX1X
// IF ?16?/0 SWITCH XXXXXXX0
//       ELSE SWITCH XXXXXXX1
// IF ?CD?/2001 IF ?2?/USER EVALUATE P2-SYSTEM
//       ELSE IF ?2?/SYSTEM EVALUATE P2-'USER
// IF ?CD?/2001 GOTO TOP
```

Figure 18-24

Prompt screen for procedure LDA

```
                             LOCAL DATA AREA. WORKSTATION  00
                                         000000
                  0---------1---------2---------3---------4---------5
                  12345678901234567890123456789012345678901234567890
    1 - 150       00000000000000000000000000000000000000000000000000
   51 - 100       00000000000000000000000000000000000000000000000000
  101 - 150       00000000000000000000000000000000000000000000000000
  151 - 200       00000000000000000000000000000000000000000000000000
  201 - 250       00000000000000000000000000000000000000000000000000
  251 - 300       00000000000000000000000000000000000000000000000000
  301 - 350       00000000000000000000000000000000000000000000000000
  351 - 400       00000000000000000000000000000000000000000000000000
  401 - 450       00000000000000000000000000000000000000000000000000
  451 - 500       00000000000000000000000000000000000000000000000000
  501 - 512       000000000000
                  12345678901234567890213456789012345678901234567890
                  0---------1---------2---------3---------4---------5

                                      S W I T C H E S
                            1    2    3    4    5    6    7    8
                            0    0    0    0    0    0    0    0
                                               CMD 1 - USER/SYSTEM
                                               CMD 7 - CANCEL IMMEDIATELY
```

Figure 18-25

SFGR specifications for LDA prompt screen

```
*        1       . 2         3         4 .       5         6         7         8
00070SSCREEN01  0124       Y                                   AG
00090D        0028 224Y                                CLOCAL DATA AREA  WORKSTX
     DATION
00110DWSID    0002 254Y  Y          Y Y
     DUORS        6 336Y  Y          Y Y     Y
     D          50 416Y                                CO----1-----2--X
     D----3-----4-----5
     D          50 516Y                                C12345678901234567890123X
     D456789012345678901234567890
     D           9 6 2Y                                C  1 -  50
00160DLINE1     50 616Y  YB                      Y
     D           9 7 2Y                                C 51 - 100
00210DLINE2     50 716Y  YB                      Y
     D           9 8 2Y                                C101 - 150
00260DLINE3     50 816Y  YB                      Y
     D           9 9 2Y                                C151 - 200
00310DLINE4     50 916Y  YB                      Y
     D           910 2Y                                C201 - 250
     D          501016Y  Y                       Y
     D           911 2Y                                C251 - 300
     D          501116Y  Y                       Y
     D           912 2Y                                C301 - 350
     D          501216Y  Y                       Y
     D           913 2Y                                C351 - 400
     D          501316Y  Y                       Y
     D           914 2Y                                C401 - 450
     D          501416Y  Y                       Y
     D           915 2Y                                C451 - 500
     D          501516Y  Y                       Y
     D           916 2Y                                C501 - 512
     D          121616Y  Y                       Y
     D          501716Y                                C12345678901234567890123X
     D456789012345678901234567890
     D          501816Y                                CO----1-----2--X
     D----3-----4-----5
     D          152034Y                                CS W I T C H E S
     D          292126Y                                C1   2   3   4   5   6  X
     D 7   8
     DSWIT1      12226Y  YN   Z             Y    Y
     DSWIT2      12230Y  YN   Z             Y
     DSWIT3      12234Y  YN   Z             Y
     DSWIT4      12238Y  YN   Z             Y
     DSWIT5      12242Y  YN   Z             Y
     DSWIT6      12246Y  YN   Z             Y
     DSWIT7      12250Y  YN   Z             Y
     DSWIT8      12254Y  YN   Z             Y
     D          402340Y                Y       Y       CCMD 1 - USER/SYSTEM    X
     D
     D          402440Y                Y       Y       CCMD 7 - CANCEL IMMEDIATX
     DELY
```

Saving and Restoring the LDA and UPSI Switches

by Mel Beckman

Code on diskette:

Procedures PUSHLDA, POPLDA
RPG programs PSHLDA, POPLDA

When trying to integrate different applications, I often discover that conflicting LDA and UPSI switch usage causes unpredictable failures. For example, Package A might use certain LDA positions for one purpose, while Package B uses the same positions for a completely different purpose. The packages might run fine separately, but when combined through a common procedure, neither package works right.

My solution — a pair of utilities, PUSHLDA and POPLDA — saves and restores the contents of the LDA and UPSI switches on a stack in a last-in, first-out (LIFO) fashion. Calling procedure PUSHLDA "pushes" the contents of the LDA and switches onto the stack, while calling procedure POPLDA "pops" them off the stack. Using a stack as a save area lets you save and restore the LDA and switch contents reliably, even in nested procedures. And because the stack is stored in a RETAIN-J disk file, which is unique for every job, you can be certain each job has its own private stack.

Programs PSHLDA and POPLDA (Figures 18-26 and 18-27, respectively) both use the direct file LDASTACK, which contains 100 512-byte records. The first 99 records make up the LDA stack. The last record — a control record — records the current depth of the stack in positions 1 and 2 and the stack of UPSI switch values in positions 256-354. The eight UPSI switches are represented as eight bits in a single byte; the UPSI stack is a 99-byte array.

Procedures PUSHLDA and POPLDA (Figures 18-28 and 18-29, respectively) each load a corresponding program — PSHLDA or POPLDA — and reference the LDASTACK file with a // FILE statement. The // FILE statement defines file LDASTACK as RETAIN-J with RECORDS-100. The first time PUSHLDA is called in a job, SSP automatically creates the LDASTACK file as a direct file. Subsequent calls in the same job to PUSHLDA and POPLDA use this same file.

When program PSHLDA runs, it retrieves the control record to get the stack depth counter and UPSI array, increments the stack depth counter, and saves the contents of the LDA in the direct file record to which the stack depth counter is pointing. It then converts the UPSI switches to bit-values in a byte, stores the byte in the UPSI array, and rewrites the control record to update the stack depth counter and UPSI array in the LDASTACK file.

When program POPLDA runs, it reverses the process: it reads the control record, decrements the stack depth counter, and updates the control record. It then reads the record to which the stack depth counter is pointing (before decrementing) to restore the contents of the LDA. Finally, program POPLDA uses the UPSI switch byte to restore the UPSI switch values.

When using procedures PUSHLDA and POPLDA, keep in mind that you always must perform push and pop operations in tandem. If you call PUSHLDA in a procedure without a later call to POPLDA, the stack will be out of synch with other procedures, resulting in interference between procedures.

Figure 18-26

Program PSHLDA

```
*        1         2         3         4         5         6         7         8
0001 H                                                                      PSHLDA
0002 F*
0003 F* Push the LDA and UPSI switches onto the LDA stack
0004 F*
0005 FLDASTACKUC  F     512R           DISK
0006 E                    UPSI        99  1
0007 ILDASTACK
0008 I                                         1 256 DATAA
0009 I                                       257 512 DATAB
```

```
I           UDS
I                                        1 256 LDAA
I                                      257 512 LDAB
C           100       CHAINLDASTACK                   Get control record
C                     MOVELDATAA     L      20        Extract stack level
C                     MOVEADATAB     UPSI,1           Extract UPSI stack
C                     ADD  1         L                Bump stack level
C*
C           L         CHAINLDASTACK                   Get stack record
C                     EXCPTLDA                        Update it
C*
C                     BITOF'01234567'UBYTE      1         Collect UPSI bits
C   U1                BITON'0'       UBYTE
C   U2                BITON'1'       UBYTE
C   U3                BITON'2'       UBYTE
C   U4                BITON'3'       UBYTE
C   U5                BITON'4'       UBYTE
C   U6                BITON'5'       UBYTE
C   U7                BITON'6'       UBYTE
C   U8                BITON'7'       UBYTE
C                     MOVE UBYTE     UPSI,L
C*
C           100       CHAINLDASTACK                   Get control record
C                     EXCPTCTRL                       Update it
C                     SETON                     LR    E.O J
OLDASTACKE                LDA
O                         LDAA     256
O                         LDAB     512
O        E                CTRL
O                         L          2
O                         UPSI     355
```

Figure 18-27

Program POPLDA

```
*        1         2         3         4         5         6         7         8
H                                                                  POPLDA
F*
F* Pop the LDA and UPSI switches off of the LDA stack
F*
FLDASTACKUC  F     512R          DISK
E                    UPSI      99  1
ILDASTACK
I                                        1 256 DATAA
I                                      257 512 DATAB
I           UDS
I                                        1 256 LDAA
I                                      257 512 LDAB
C           100       CHAINLDASTACK                   Get control record
C                     MOVELDATAA     L      20        Extract stack level
C                     MOVEADATAB     UPSI,1           Extract UPSI stack
C           L         SUB  1         N      20        New stack level
C                     EXCPTCTRL                       Update control rec
C*
C           L         CHAINLDASTACK                   Get stack record
C                     MOVE DATAA     LDAA             Move to LDA
C                     MOVE DATAB     LDAB
C*
C                     MOVE UPSI,L    UBYTE      1         Get UPSI bits
C                     TESTB'0'       UBYTE         U1 Set UPSI switches
C                     TESTB'1'       UBYTE         U2
C                     TESTB'2'       UBYTE         U3
C                     TESTB'3'       UBYTE         U4
C                     TESTB'4'       UBYTE         U5
C                     TESTB'5'       UBYTE         U6
C                     TESTB'6'       UBYTE         U7
C                     TESTB'7'       UBYTE         U8
C*
C                     SETON                     LR    E 0.J
OLDASTACKE                CTRL
O                         N          2
```

Figure 18-28
Procedure PUSHLDA

```
* Push the LDA and UPSI switches onto the LDA stack
// LOAD PSHLDA
// FILE NAME-LDASTACK,RECORDS-100,RETAIN-J
// RUN
```

Figure 18-29
Procedure POPLDA

```
* Pop the LDA and UPSI switches off of the LDA stack
// LOAD POPLDA
// FILE NAME-LDASTACK,RECORDS-100,RETAIN-J
// RUN
```

Granting Console Capability to Any Workstation

by Mel Beckman

Code on diskette:
Procedures GOLEM, ROLEM
Assembly language programs GOLEM, ROLEM

To properly manage access to your S/36 resources, you need to weigh the user's "need to know" against the user's potential for causing damage. Thus, SSP restricts nonconsole workstation operators from viewing and changing spool and job queue entries for other users and restricts access to active jobs and to certain console commands. At the same time, SSP provides a way to grant console operational capabilities to a workstation other than the system console: the System Service Device (SSD). The System Service Device is allowed access to the same commands as the system console, except for the ASSIGN, STOP SYSTEM, and VARY commands. The system console operator can give SSD privileges to only one workstation at a time by using the START SERVICE command.

However, in certain situations, you may want more than one user to have SSD status or you may want to give users SSD status without bothering the system operator. The following pair of small assembly language programs do just that.

The first program, GOLEM (Grant Operational Liberty for Everything Meaningful), turns on SSD privileges, and the second, ROLEM (Revoke Operational Liberty for Everything Meaningful), turns them off. In addition, GOLEM sets the CONSOLE GIVE flag for a workstation, allowing it to acquire the console unilaterally via the CONSOLE TAKE command (without the system operator issuing a CONSOLE GIVE command manually).

The procedures in Figures 18-30 and 18-31 run GOLEM and ROLEM, respectively. Security officer authority is required to run the MKGOLEM procedure, and Service Aids authority is required to run GOLEM or ROLEM.

Figure 18-30
Procedure GOLEM

```
* Grant Operational Liberty for Everything Meaningful
// LOAD GOLEM
// RUN
```

Figure 18-31

Procedure ROLEM

```
* Revoke Operational Liberty for Everything Meaningful
// LOAD ROLEM
// RUN
```

Re-creating Programs GOLEM and ROLEM

If you don't have assembler routines GOLEM and ROLEM, you can re-create them with procedure MKGOLEM (you don't need IBM's Assembler Language Program Product to install GOLEM and ROLEM). To run MKGOLEM, you must be signed on as a security officer, and the system must be dedicated.

```
*
* This procedure makes the GOLEM and ROLEM programs
*
*
*- First, make a copy of a dummy load members to clone
*
// LOAD $MAINT
// RUN
// COPY FROM-#LIBRARY,TO-#LIBRARY,LIBRARY-O,NAME-#FECPY,NEWNAME-GOLEM
// COPY FROM-#LIBRARY,TO-#LIBRARY,LIBRARY-O,NAME-#FECPY,NEWNAME-ROLEM
// END
*
* Now patch in GOLEM's genetic material
*
// LOAD $FEFIX
// RUN
HDR
PTF OGOLEM,,#LIBRARY
DATA F2,0000,35A1 1018 75A1 0075,A117 75A1,C47A,4075,00
DATA 00,0010,7AA0 81F4 0004 0400,0897
END
*
* Then patch in ROLEM's genetic material
*
// LOAD $FEFIX
// RUN
HDR
PTF OROLEM,,#LIBRARY
DATA F2,0000,35A1 1018 75A1 0075,A117 75A1,C47B 0075,00
DATA 00,0010,79A0 81F4 0004 0400,0897
END
```

Running CACHE from Other Than the System Console

by Gary T. Kratzer

Code on diskette:
Procedures CADD, CREM

On the S/36, the SSP lets you run the CACHE procedure only from the system console. Also, you cannot run CACHE from the JOBQ or EVOKE it, even if the initiating workstation is the system console. This restriction is often inconvenient, especially when you dial into your S/36 from a remote site (such as your home) to perform "housekeeping" and you cannot, or don't wish to, acquire the system console.

The Cache patch (developed under SSP Release 5.1, PTF level 3705) lets you EVOKE the CACHE procedure or execute it from any workstation or the JOBQ. The patch may be applied to earlier or subsequent PTF levels if message SYS-3330 — "Checkbyte in data statement incorrect or missing" — does not occur when you attempt to apply it. Figures 18-32 and 18-33 contain the data to apply and remove the patch, respectively.

Figure 18-32

Patch to add functions to CACHE. (This procedure appears as procedure CADD on diskette.)

```
// LOAD $FEFIX
// RUN
HDR
PTF O#SVCMG,,#LIBRARY
DATA 75,0213,F20014
END
```

Figure 18-33

Code to remove patch. (This procedure appears as procedure CREM on diskette.)

```
// LOAD $FEFIX
// RUN
HDR
PTF O#SVCMG,,#LIBRARY
DATA F2,0213,75A104
END
```

Explanation of Task Work Area (#SYSTASK File)

answered by Mel Beckman

Q I would appreciate some comments on the S/36 Task Work Area (TWA) when using ASNA's RPG III or BPS's RPG II 1/2. I am working on a system with many subprograms, and my total used storage approaches 800 K. My understanding is that the unused programs are "swapped out" to TWA until needed. However, if 10 people are using the system, is the potential amount of "swapped out" programs 8 MB? Does this start creating problems for the TWA as far as the size of the TWA is concerned?

A Your assumptions are all correct: unused programs are paged out (swapped out) to the TWA (#SYSTASK file) by the S/36 virtual storage mechanism. Thus, you need a TWA large enough to hold all your activated programs for all users. I suspect you're using the "menu program" approach to hold the activations open for your most frequently used programs. This is an excellent technique and results in very good response time and much less CPU resource utilization because fewer initiations and terminations occur; you do, however, pay a price in disk space that must be reserved to hold the activated programs.

And that's where the S/36 has a slight wrinkle. Although IBM embedded the external program call mechanism into S/36 microcode, IBM never expected customers to figure that out and start using it. Thus, they never thought that the TWA would need to be very large, and they set the configurable limit at 6,553 blocks (about 16 MB). If you don't need more than 16 MB, setting the TWA size in your configuration to the maximum value will work fine. You should probably do a COMPRESS FREEFLOW and an IPL to get the TWA created adjacent to #LIBRARY — that will improve performance a bit.

But what if you need more than 16 MB? When the TWA fills up, SSP tries to expand it in noncontiguous extents using the following algorithm: the first extent is 400 blocks; the second 800; the third 1,600; the fourth 3,200; and so on. Unfortunately, it's hard to keep that much free space in one place on the disk when your system has been running for a few hours.

The trick is to "pre-activate" all of your programs at the start of the day so the TWA gets expanded before disk space becomes fragmented. You do this by planting a special "initialization" code in the parameter list for your called programs (i.e., the mainline applications that your menu program calls). Your menu program, when it first starts up, then calls each subprogram in turn (passing the initialization code). Upon seeing the initialization code, the subprograms return to the driver program immediately, without performing any I/O. Thus, the subprograms get activated (e.g., loaded and files opened) and have the TWA space allocated before any disk fragmentation occurs. Subsequent calls to your application programs are very fast because they remain activated and resident in virtual storage.

Explanation of SMF's Swap-in Counter

answered by Mike Patton

Q The other day I was using SMF to investigate the performance of our S/36, and I found a very strange thing. During one part of the day, our S/36 was running only one job, which took 64 K of our available 454 K. The strange thing was that during this same period the swap-in counter on the SMF report showed 20 swaps per minute. I don't get it! How can a 64 K program swap in and out 20 times a minute when there is plenty of memory available?

A Swapping an entire program in and out of memory (due to limited available memory) is only one reason the swap-in counter is incremented. Many of the internal system routines the S/36 uses to perform tasks (such as diskette data compression) are not part of the memory resident system supervisor. They are called transient routines, and when your program needs one of them, it is loaded from the system library, and the swap-in counter is incremented by one. The swap-in counter also is incremented for the "pseudo" swap that occurs when your program is first loaded into memory. Therefore, the twenty swaps per minute that you registered with your 64 K program does not indicate that the program was being swapped in and out that many times.

Improving on the DATAF1 Conditional Statement

by Ron Elliott

program by Matthew Henry

Code on diskette:
Procedure WHICH
RPG program WHICH

When you want to access your S/36 disk Volume Table of Contents (VTOC), assembly language subroutine SUBRVR (*Displaying the VTOC Graphically,* page 612) is quite useful. Utility WHICH, however, goes one step beyond to make full use of subroutine SUBRVR. Program WHICH accepts an object name from any calling procedure and employs subroutine SUBRVR to return a wealth of useful VTOC information that the calling procedure can interrogate and act upon.

If the object is a data file, program WHICH tells you the file organization (indexed, sequential, or direct), the record length, and the key data for indexed files. You can use this information when a procedure prompts you for a file name used in program processing. You also can use program WHICH to verify that the user-supplied file name does exist and that the record length and key data are as expected before the program starts to process data. Traditionally, the // IF DATAF1-object name statement alerts the user that the object, ***but not its type,*** exists. Program WHICH, however, tells you what the specific object type is (i.e., file, folder, or library). In addition, program WHICH works well with IDDU/QUERY functions to determine whether the specified file is linked to IDDU. Finally, the VTOC data returned by program WHICH comes in handy for deleting parent files with alternate indexes and for getting at the key data for indexed files with a minimum of fuss.

How Program WHICH Works

Utility WHICH consists of procedure WHICH (Figure 18-34), program WHICH (Figure 18-35), and subroutine SUBRVR. You call program WHICH by calling procedure WHICH and specifying the object name for which you want information as the first parameter. Program WHICH passes the object name to the assembler subroutine through the VTOCDS data structure. The VTOC data obtained by the subroutine then is returned to VTOCDS from subroutine SUBRVR. Next, program WHICH begins to examine the data structure and sets up UPSI switches and LDA fields based upon the VTOCDS contents.

If field FFORG (file organization) is blank, external indicator U1 is set on to signal procedure WHICH that the given object does not exist in the VTOC. If the given object does exist, program WHICH uses a series of CAS commands to execute internal subroutines that set other UPSI switches that define what kind of object the given item is. The comments in lines 15 through 24 of Figure 18-35 describe the external indicators that pass information to the procedure. Note that the program uses a TESTB (test bits) instruction to examine the second byte of the returned SSP attributes (FFATB2) to set on indicator U8 if the object file is currently linked to IDDU. If indicator U8 is on in your procedure, it helps you avoid an error message resulting from an attempt to link an already linked file.

Lines 100 through 158 move data file information into the LDA data structure. After line 158 is executed, the UDS data structure contains detailed file information as described by the program comments. Note that program WHICH moves the literal MULT to the field labeled LDKEYP if an indexed file has a multipart key. The partial key lengths and positions are moved into the fields labeled LDKYL1 (partial key length 1), LDKYP1 (partial key position 1), and so on. If the indexed file doesn't have a multipart key, these fields remain blank.

Fields LDPARN, LDALTS, and LDXTND, the last three items in the data structure, are quite valuable. If the given object is an alternate index, field LDPARN contains the parent file name. If the given object is a parent file, field LDALTS contains PARENT to indicate the existence of alternate indexes. Because a parent file cannot be deleted until all alternate indexes are deleted, this parameter saves time identifying parent files.

Because program WHICH uses the UPSI switches and places data into the LDA, the data it retrieves is available for subsequent processing in the calling procedure. Therefore, you can code your procedure to return a message to the operator, replace an existing EXTEND parameter (armed for a record-adding program), delete an alternate index before attempting to delete a parent file, skip over a link to IDDU if the file is already linked, ensure that a subsequent program is coded with the proper key information, or whatever else you need to find out in advance of processing.

As you can see, program WHICH is quite useful just the way it is presented here, but with modifications such as those mentioned above, you can build in many functions for your own environment. Whatever its use, program WHICH can tell you which, what, and when, but, being a computer function, it can't tell you why.

Figure 18-34

Procedure WHICH

```
// SWITCH 00000000
// LOCAL OFFSET-1,BLANK-54,AREA-USER
// LOAD WHICH
// RUN
NAME-?1R?
// RETURN
********************************************************************
*       Name  WHICH
```

```
*    Purpose: Report information from the VTOC on the NAME
* Parameters: P1 - VTOC label
*
*   Switches: U1 - Label does not exist in the VTOC
*             U2 - Indexed file
*             U3 - Sequential file
*             U4 - Direct file
*             U5 - Library
*             U6 - Folder
*             U7 - Alternate index file
*             U8 - Linked to an IDDU definition
*
*        LDA: 001-004 - Record length for a file
*             005-010 - Creation date of label
*             011-013 - Key length   (- total length if multi-part index)
*             014-017 - Key position (- MULT if multi-part index)
*             018-020 - Key length 1 for multi-part indexes
*             021-024 - Key position 1
*             025-027 - Key length 2
*             028-031 - Key position 2
*             032-034 - Key length 3
*             035-038 - Key position 3
*             039-046 - Parent file name if alternate index
*             047-054 - Extend value for a file
*             055-060 - PARENT if file is a parent file (alternates attached)
```

Figure 18-35

Program WHICH

```
* . . 1 . . . . 2 . . . . . 3 . . . . 4 . . . 5 . . . 6 . . . 7 . . .    8
H*************************************************************************
H* Program. WHICH   Written by  Matthew P  Henry                          *
H*                                 SUBRVR supplied by Mel Beckman         *
H* This program accepts a resource name from the calling procedure*
H* and sets on USPI switches indicating what type the resource is *
H*                                                                        *
H* The record length and creation date are returned in the LDA    *
H* The key lengths and positions are also returned in the LDA     *
H* This program tests only for local files   No remote files are  *
H* dealt with.  It also will work only on a System/36             *
H*---------------------------------------------------------------*
H* Indicators                                                     *
H* None                                                           *
H*************************************************************************
H* UPSI Switches                                                  *
H* U1      - On -Label doesn't exist on the system                *
H*           Off-Label does exist                                 *
H* U2      - On -Label is an indexed file                         *
H* U3      - On -Label is a sequential file                       *
H* U4      - On -Label is a direct file                           *
H* U5      - On -Label is a library                               *
H* U6      - On -Label is a folder                                *
H* U7      - On -Label is an alternate indexed file               *
H* U8      - On -File is linked to IDDU                           *
H*---------------------------------------------------------------*
H* LDA data.                                                      *
H* 001-004- Record length for a file                              *
H* 005-010- Creation date                                         *
H* 011-013- Key length   (- total length if multi-part index)     *
H* 014-017- Key position (- MULT if multi-part index)             *
H* 018-020- Key length 1 for multi-part indexes                   *
H* 021-024- Key position 1                                        *
H* 025-027- Key length 2                                          *
H* 028-031- Key position 2                                        *
H* 032-034- Key length 3                                          *
H* 035-038- Key position 3                                        *
H* 039-046- Parent file name if alternate index                   *
H* 047-054- Extend value for a file                               *
H* 055-060- PARENT if file is a parent file (Alts attached)       *
H*************************************************************************
H      64                                                          WHICH
FINFOFILEID     120 120            SPECIAL       SUBR01
IINFOFILENS
I                                      6  13 LABEL
IVTOCOS      DS
I                                      1   1 FFORG
```

```
I                                           3  10 FFLABL
I                                          11  16 FFCRDT
I                                          20  20 FFIXFG
I                                          22  22 FFATB2
I                                          41  44 FFRECL
I                                          54  56 FFKEYL
I                                          57  60 FFKEYP
I                                          69  76 FFXTND
I                                          89  96 FFIDDU
I                                          98 100 FFKYL1
I                                         101 104 FFKYP1
I                                         105 107 FFKYL2
I                                         108 111 FFKYP2
I                                         112 114 FFKYL3
I                                         115 118 FFKYP3
I                                         119 126 FFPARN
I            DS
I                                           1   60YYMMDD
I                                           1   20YY
I                                           3   80MMDDYY
I                                           7   80YY2
I           UDS
I                                           1   4 LDRECL
I                                           5  10 LDCRDT
I                                          11  13 LDKEYL
I                                          14  17 LDKEYP
I                                          18  20 LDKYL1
I                                          21  24 LDKYP1
I                                          25  27 LDKYL2
I                                          28  31 LDKYP2
I                                          32  34 LDKYL3
I                                          35  38 LDKYP3
I                                          39  46 LDPARN
I                                          47  54 LDXTND
I                                          55  60 LDALTS
C                     READ INFOFILE                    Get info from proc.
C                     EXIT SUBRVR                      Call subroutine
C                     RLABL          LABEL
C                     RLABL          VTOCDS
C           FFORG     IFLE ' '                         File exist?
C                     SETON                     U1
C                     ELSE
C           FFORG     CASEQ'I'       SIDX              Indexed
C           FFORG     CASEQ'S'       SSEQ              Sequential
C           FFORG     CASEQ'D'       SDIR              Direct
C           FFORG     CASEQ'L'       SLIB              Library
C           FFORG     CASEQ'F'       SFLD              Folder
C           FFORG     CASEQ'X'       SALT              Alternate index
C                     END
C                     END
C*
C                     TESTB'6'       FFATB2         U8 IDDU linked?
C*
C  NU1                DO                             Do is file exist
C     NU5NU6          DO                             Do if file
C                     MOVE FFRECL    LDRECL            Move recl to LDA
C                     END
C                     MOVE FFCRDT    YYMMDD            Move creation
C                     MOVE YY        YY2                date to LDA in
C                     MOVE MMDDYY    LDCRDT             MMDDYY format
C                     MOVE FFKEYL    LDKEYL            Key length
C           FFKEYP    IFGT '4096'                      Key length > max record length
C                     MOVE 'MULT'    LDKEYP            Must be multi part
C                     ELSE
C                     MOVE FFKEYP    LDKEYP            Key position
C                     END
C                     MOVE FFKYL1    LDKYL1            Multi-key length 1
C                     MOVE FFKYP1    LDKYP1                    position 1
C                     MOVE FFKYL2    LDKYL2                  key length 2
C                     MOVE FFKYP2    LDKYP2                    position 2
C                     MOVE FFKYL3    LDKYL3                  key length 3
C                     MOVE FFKYP3    LDKYP3                    position 3
C                     MOVE FFPARN    LDPARN          Alternate's parent
C                     MOVE FFXTND    LDXTND          File extend value
C                     TESTB'1'       FFIXFG        10Test Index flag
```

```
C   10              MOVE 'PARENT'  LDALTS
C  N10              MOVE '      '  LDALTS
C                   END
C                   SETON                          LR End program
C*****************************************************************
C*                    SUBROUTINES                                *
C*****************************************************************
C* Indexed file
C          SIDX     BEGSR
C                   SETON                     U2
C                   ENDSR
C*
C* Sequential file
C          SSEQ     BEGSR
C                   SETON                     U3
C                   ENDSR
C*
C* Direct/relative file
C          SDIR     BEGSR
C                   SETON                     U4
C                   ENDSR
C*
C* Library
C          SLIB     BEGSR
C                   SETON                     U5
C                   ENDSR
C*
C* Folder
C          SFLD     BEGSR
C                   SETON                     U6
C                   ENDSR
C*
C* Alternate indexed file
C          SALT     BEGSR
C                   SETON                     U7
C                   ENDSR
C*
```

Sending a Message to the Console

by Michael J. Ranks

Code on diskette:

Assembler subroutine SUBRFD

If you're running a NEP (Never-Ending Program) and want to bring a condition that has arisen to the attention of the system operator, it would be handy to be able to do so from within that program. The subroutine SUBRFD makes such communication possible by sending messages to the system console from within an executing RPG program.

Subroutine SUBRFD sends a 75-character message to the console whenever it is called. To call the subroutine from an RPG program, code the following anywhere your program logic might encounter a condition the system operator needs to be alerted to:

```
C                         EXIT SUBRFD
C                         RLABL          CONFLD 75
```

The field name in the RLABL statement contains the message you want displayed on the console. It must be an alphabetic field (not an array or array element) exactly 75 characters long. I use a table to store the messages and put the time and date at the end of each message. Then I move the data to the field before executing the subroutine.

Re-creating Subroutine SUBRFD

If you don't have assembler subroutine SUBRFD, you can re-create it with procedure MKSUBRFD (you don't need IBM's Assembler Language Program Product to install SUBRFD). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRFD. You need to run MKSUBRFD only once to create the SUBRFD subroutine.

```
// * 'Re-creating R module SUBRFD in library #RPGLIB'
* Build an empty member in a #MAINT file with the correct directory entry
// LOCAL OFFSET-201 DATA-'00000068'     Number of #MAINT records
// LOCAL OFFSET-209 DATA-'
'D9E2E4C2D9C6C4404000000040000000000000'00000000990002000000000000
// LOCAL OFFSET-273 DATA-'
'0000000000000000E217E214C2D9C6C4000000000F3000284
// LOAD MAKMEM
// FILE NAME-BINARY LABEL-#MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD #MAINT
// FILE NAME-#MAINT RETAIN-S
// RUN
// COPY FROM-DISK,FILE-#MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRFD member to insert object code
// LOAD $FEFIX
// RUN
HDR  3858 SUBRFD0000
PTF  2847 RSUBRFD,99,,#RPGLIB
DATA 9814 00 000C C7D9C6C4010000000000000000000000000000000000000000000000000000
DATA 0748 00 002C 00000000000000000000000000000000E32F002F3408000234010084340200B8
DATA 81B3 00 0040 3502008220010008B03CFC1008B008A350200862C4A00904BC2C2003EF4010406
DATA 8AE3 00 006C 0F0100820002F77721E1A1813DF0B0703F31F000450080C360100843502008035 10
DATA AF1B 00 008C 0082030000248000400000000000000000000000000000000000000000000000
DATA 3070 00 00AC 00000000C0000000000000001300080601E33A000300048000340C36040F1F9F0F9
DATA 4086 00 00CC 404007D9C8C3C9E2C8080840E70BC8E3E8C1D9C840C9D6F3C509C8C1C3C5E2B8
DATA E180 00 00EC 40C9D5C34BC4C9E20703C1E840C14000E31E0DF2F7F5C240C8C9C503C440C6D6
DATA 77EC 00 0100 40C3D6D5E2D8D3C540C1C6C4400BC5E3E40905000000000000000000000000000
DATA 8898 00 012C 000000000000000000000000000000C6FFFFF2000000000000000000000000000
DATA E783 00 0140 0000000000000000000000000000000000000000000000000000000000000000
DATA 3856 00 0160 0000000000000000000000000000000081SCD000000000000000000000000000
DATA EF28 00 0180 0000000000000000000000000000000000000000000000000000000000000000
DATA 85EE 00 01A0 0000000000000000000000000000000040404040404040404040404040404040
DATA 7488 00 01C0 4040404040404040404040404040404040404040404040404040404040404040
DATA 8384 00 01E0 4040404040404040404040404040404040404040404040404040404040404040
END  5608
```

Creating Console Messages That Survive an IPL

answered by Georgia Agallianos

Q We run unattended utilities on our S/36 at night and would like to send messages to the console that the procedures were run successfully or aborted. When we IPL each morning, messages sent to the console are erased if we use // MSG or // * statements. If we use the // ** statement, we can't IPL the system. How can we leave messages for a console operator (there are several operators) that will allow and survive an IPL?

A The system won't let you retain messages sent to a specific workstation if an IPL has occurred since the message was sent. You can, however, send a message to a specific user ID and have it survive an IPL. This method is possible only if the targeted user is not signed on to the system at the time the message is sent. If the user is signed on, the message is displayed when the OFF command is executed. You could create a "dummy" user ID for this purpose, and, after IPLing, have the system operator sign on with this ID to check for messages.

Outputting to SYSLIST Device

answered by Mike Patton

Code on diskette:

Assembler subroutine SUBRSY

Q Is there a way on the S/36, from within RPG II, to access the device-independent SYSLIST function?

A Several commercially available routines perform the function you desire. I also wrote one such routine that is in the COMMON Graffiti library in public domain. To access this routine from an RPG program, simply code the following C-specs:

```
C                              EXIT SUBRSY
C                              RLABL          DATA   132
```

Note in these specifications that DATA (the name I have arbitrarily chosen for the field) has a defined length of between 1 and 132, the latter being the maximum SYSLIST output record size.

Re-creating Subroutine SUBRSY

If you don't have assembler subroutine SUBRSY, you can re-create it with procedure MKSUBRSY (you don't need IBM's Assembler Language Program Product to install SUBRSY). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRSY. You need to run MKSUBRSY only once to create the SUBRSY subroutine.

```
// * 'Re-creating R-module SUBRSY in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C209E2E84040000000400000000000000102000200990002300000000000'
// LOCAL OFFSET-273,DATA-+
'00000000000000000803F9900000000000000000000C804000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
```

Continued

```
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRSY member to insert object code
// LOAD $FFIX
// RUN
HDR  38C8 SUBR500000
PTF  0CF8 RSUBRSY,99,,#RPGLIB
DATA CB44 00 0000 E20BE2E4C309E2E8000000000C7300000000000000000000000000000000000
DATA 6767 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 2RB0 00 0040 F331D031F38710BCF7F4C704F2F440F1F1B1F1B1F8F54C340100BC340200B034
DATA 00A2 00 0060 08D0643880008CF2\0203501008 40F01C0080089 1D0000000002F2028221E1A18
DATA 7186 00 0080 E3800042008001C00006FC01C010072020E00008FC0470F0100720C08C20200
DATA 5188 00 00A0 8AF401040808E010084008 8C2010000C2020000C087002624 1C181B1210080801
DATA 3648 00 00C0 E30F0072000000030100004001020000008000000000000000000000000000000
DATA 5F0E 00 00E0 0000000000000000000000000000000000000000000000000000000000000000
DATA 6E48 00 0100 C5FFFF7200000000000000000000000000000000000000000000000000000000
DATA F715 00 0120 0000000000000000000000000000000000000000000000000000000000000000
DATA 6E87 00 0140 B14C000000000000000000000000000000000000000000000000000000000000
DATA 062E 00 0160 0000000000000000000000000000000000000000000000000000000000000000
DATA 00F6 00 0180 4040404040404040404040404040404040404040404040404040404040404040
DATA A2A1 00 01A0 004000007F800000000000000020000020051C0030000000000000000000000000
DATA 3019 00 01C0 00803F880000000000000000087C07C7F4F140404000C7F9000000000000000200
DATA 468F 00 01E0 000200B1000030000000000000000000000000000803FC0000000000000000000FF
END  F8C8
```

Using Autoresponse for Specific Messages

answered by Gary T. Kratzer

Q I am using a protocol converter to set up a S/36 remote inquiry system that will be used at night when the S/36 is unattended. I need a way to avoid "hanging" sessions until morning if a call is abnormally disconnected. Unfortunately, message SYS-7300 (Display Station Not Communicating with System) has a severity level of 5, so autoresponse is not available. If not answered, the interrupted session holds open the inquiry files and halts my automated "night" routine. Is there a way around this problem?

A You can change the severity level, as well as the autoresponse value, for any message. First, you must create an autoresponse source member. The format of the response specification statement is:

```
MIC  response, severity level    (comment)
```

Valid responses are 0, 1, 2, 3, and D (dump); valid severity levels are 1, 2, 3, 4, and 5. Assuming you want to answer the SYS-7300 message with option 2 (when it's available), your autoresponse member could look something like this:

```
SYS
7300  2,1    Display station not communicating
```

This code assigns default option 2 and severity level 1 to the message. Note that only messages with a severity level equal to or less than the severity level (specified with the // NOHALT statement) for the system, session, or your job are answered automatically. For more information, see chapter 14 in the *S/36 Concepts and Programmer's Guide* (SC21-9019).

Displaying System Error Message Text

by Victor J. Vergata

Code on diskette:
Procedure SYSERR

I frequently receive calls from S/36 users who have a system error with options, and either the users are not reading the text of the message completely, or they don't know what the available options do. To give the users additional help with less input from me, I created procedure SYSERR:

```
// MEMBER USER1-##MSG1,USER2-##MSG4,LIBRARY-#LIBRARY
// ERR ?1R?
```

This procedure accesses the IBM level 1 and level 2 message members to display the text of a system message. The user can press the Enter key to display available additional help text that explains the options. This procedure is most helpful with spool-related messages. To invoke the procedure, key:

```
SYSERR nnnn
```

where *nnnn* is the four-digit error number. If you omit the *nnnn*, the procedure prompts "Enter missing parameter."

Retrieving the CPU Serial Number

by Mel Beckman

Code on diskette:
Assembler subroutine SUBR##

An assembler language subroutine that reads the S/36 machine serial number is a useful routine because it lets RPG programmers write programs that run only on machines with certain serial numbers, thus affording protection against software theft. Subroutine SUBR## provides this capability.

To use subroutine SUBR##, code the two statements below into your RPG program wherever you want to check the serial number:

```
C                           EXIT SUBR##
C                           RLABL          SERIAL  6
```

When these two statements are executed, the machine serial number will be placed in field SERIAL, which must be six characters long. The RPG program then can test SERIAL against a predetermined constant to ascertain whether the program should run on the system from which the serial number was retrieved.

Incidentally, the memory location of the serial number (000898 hex) has been documented by IBM in the *S/36 System Data Areas* manual. The machine serial number is set at the factory. If a programmer expects his software to run on machines outside the United States, I recommend that only the rightmost five digits of the serial number be used. I have discovered that the first digit is sometimes blank for machines outside the United States.

Re-creating Subroutine SUBR##

If you don't have assembler subroutine SUBR##, you can re-create it with procedure MKSUBR## (you don't need IBM's Assembler Language Program Product to install SUBR##). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBR##. You need to run MKSUBR## only once to create the SUBR## subroutine.

```
// * 'Re-creating R-module SUBR## in library #RPGLIB
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000038      Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D97B7B4040000000000000000000001420002009000130000000000'
// LOCAL OFFSET-273,DATA-+
'0000000000000000030890000000000000000000C8040000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBR## member to insert object code
// LOAD $PEFIX
// RUN
HDR  383A SUBR#00000
PTF  CF83 RSUBR##,99,,#RPGLIB
DATA 188A 00 0000 E2D8E264C2D97B7B0000000C8800000000000000000000000000000000000000
DATA A24D 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 638E 00 0040 E321C0213402001A34080008C2020000890202F8008C8C050000[illegible]
DATA A15F 00 0060 008EF087039C01370S8C01540B8CA00030C8000F28103[illegible]
DATA 839F 00 0080 E32EC0670000F401040F8CF062800008F2810C8CF152B8[illegible]
DATA 6887 00 00A0 00000CAE018408C2010000C2020000008700000000000000000000000000000000
DATA 6688 00 00C0 C6000050177730[illegible]
DATA 7798 00 00E0 0000040062C00000600000000000000000000000000000000000000000000000
END  29EF
```

Determining the System Date Format

answered by Mel Beckman

Code on diskette:

Assembler subroutine SUBRDF

Q Our programs run in several different countries that use different conventions for the date format. We need a way to determine the system date format (MDY or DMY) within an RPG program so that we can make our programs country-independent without recompiling them. Do you have any suggestions?

A S/36 SSP allows a user to retrieve the system date format by using the $INFO assembly language macro. A small assembly language subroutine, SUBRDF, is used to allow RPG programs to make this request. To call SUBRDF, use the following code:

```
EXIT SUBRDF
RLABL           DATFMT  1
```

The RLABL is a one-character field that is set to M if the date format is MDY, D for DMY, and Y for YMD. Note that SUBRDF returns the date format for the current session. If a SET command is issued to change the format, the value from the SET command is returned.

Re-creating Subroutine SUBRDF

If you don't have assembler subroutine SUBRDF, you can re-create it with procedure MKSUBRDF (you don't need IBM's Assembler Language Program Product to install SUBRDF). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code*, page 38) to run MKSUBRDF. You need to run MKSUBRDF only once to create the SUBRDF subroutine.

```
// * 'Re-creating R-module SUBRDF in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000039'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D9E2E4C2D9C4C640400000002000000000000001020002009800013000000000000'
// LOCAL OFFSET-273,DATA-+
'00000000000000000803FCC0000000000000000000C804000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-26,EXTEND-26
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRDF member to insert object code
// LOAD $FEFIX
// RUN
HDR  3850 SUBRD00000
PTF  185E RSUBRDF,99,,#RPGLIB
DATA 74C3 00 0000 E2D8E2E4C2D9C4C60000000000680000000000000000000000000000000000000000
DATA FD88 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA C400 00 0040 E3300030340800343401002C340200303501003 41C010038020F010038003FC2
DATA 43FC 00 0060 020035F401040F0E0100340030C2010000C2020000002828 1E1A18130F080703
DATA F520 00 0080 E30E003FC087000000010000000000000003000000000052895006F210038CF2523C
DATA E642 00 00A0 000000AE016408C2010000C2020000C087000000000000000000000000000000
DATA E611 00 00C0 C500005C17773E7C40FF5C88FEFF751018F00000F0000030000068000000620C
DATA 0A1C 00 00E0 0000640062C000006000000000000000000000000000000000000000000000000
END  8873
```

Retrieving the System Date in a Procedure

answered by Matthew Henry and Gary T. Kratzer

Q I want to create a procedure that waits until a particular time to evoke a communications job. I'm using a // WAIT statement with a maximum value of 24 hours and a counter for *X* number of days. The problem is that at 2400 hours, the system date is updated, but the session and program dates are not. The ?DATE? OCL substitution value returns only the session date, not the system date. How can I retrieve the system date in my procedure?

A You can write a simple RPG program to retrieve the system date using the TIME operation code and then store the resulting system date in the LDA where your procedure can access it. Refer to *Programming with RPG II* (SC21-9006) for more details.

Resetting the System Time Without IPL

by Michael K. Maenhout

Code on diskette:
Procedure SYSTM
Assembly language program SYSTM

It *is* possible to reset the system time on the S/36 without having to IPL! A patch can create program SYSTM, which sets the system time with a value retrieved from positions one through six of the local data area. Procedure SYSTM (Figure 18-36) can be used to run program SYSTM to reset the time of day.

The assembly language source program (Figure 18-37) also is given for those who want to know how SYSTM works internally. The control storage transient scheduler SVC (X'50') is documented in the *Functions Reference Manual* (SA21-9436) as capable of returning the system time when calling transient ID X'OA' (interval timer master). Although the documentation indicates only how to retrieve the time using an in-line parameter of X'40', the system time can be set by using an in-line parameter of X'00'. The change from X'40' to X'00' lets a privileged program set the system time from the timer request block (TRB) instead of having the system time returned to the TRB.

Testing shows that // WAIT statements still operate correctly using either the INTERVAL or TIME option.

Figure 18-36

Procedure SYSTM

```
// * 'THE CURRENT TIME IS ?TIME? '
// * 'ENTER THE NEW SYSTEM TIME IN THE FORM  HHMMSS '
// EVALUATE P1,6-000000?R?
// LOCAL OFFSET-1,DATA-'?1?'
// LOAD SYSTM
// RUN
// PAUSE 'TIME HAS BEEN SET TO '?TIME?'
// RETURN
```

Figure 18-37

Program SYSTM

```
****
* SYSTM    .M.MAENHOUT
* RESETS THE SYSTEM TIME
****
XR2      EQU   2
SVCCXNT  EQU   X'50'
SYSTM    START X'1800'
         $INFO PLIST-LDA                  MOVE LDA BYTES 1-6 TO ZTIME
         MVC   TIMDAT(1) FUNC             TIME IS DECIMAL
         LA    TIMDAT.XR2                 POINT TO TIMER REQ  BLOCK
         SVC   SVCCXNT 00                 CONTROL STORAGE XIENT
         DC    XL3 0A0000'                TIME SET OPTION
EOJ      EQU   *
         $EOJ
FUNC     DC    XL1 08'
TIMDAT   $TRB  V-ALL
         $INFO
LDA      $INFO GET-LOCUSER,BUFFER-ZTIME,LEN-6 OFFSET-1
*
ZTIME    EQU   TIMDAT+$TRBTIME 5
         END
```

Re-creating Program SYSTM

If you don't have assembler routine SYSTM, you can re-create it with procedure MK SYSTM (you don't need IBM's Assembler Language Program Product to install SYSTM). To run MKSYSTM, you must be signed on as a security officer, and the system must be dedicated.

```
**
** CREATE THE LOAD MODULE SYSTM TO BE PATCHED.
**
// LOAD $MAINT
// RUN
// COPY FROM-#LIBRARY,TO-#LIBRARY,LIBRARY-O,NAME-$CNHLP,NEWNAME-SYSTM,RETAIN-R
// END
**
** APPLY PATCH
**
// LOAD $FEFIX
// RUN
HDR
PTF OSYSTM.MKMLIBR
DATA F2,0000,C202,182B,F401,040F,0C00,181D,181C,C202,181D
DATA 18,0010,181D,F400,600A,0000,F401,0404,0800,0000,0000
DATA 00,0020,0000,0000,0000,0000,0000,0000,8918,1F00,0000
DATA 00,0030,0000,0006,0000
END
```

Changing Session Dates When System Date Was Changed Without IPLing

answered by Matthew Henry and Mike Patton

Q Your procedure to change the system date without re-IPLing was invaluable. I noticed, however, that if our system is not shut down or IPLed over a 24-hour period, the system date rolls correctly but the session date does not. For example, if I IPL the system on Monday morning, the session date matches the system date, and all files created that date have the correct File Created date. But on Tuesday, the session date still reflects Monday's date, and files created from that workstation retain Monday's file creation date. Why? And what can be done to correct the dates?

A The session date changes only when the session ends or a // DATE statement is executed. A session ends only when you sign off. If you want the actual system date to appear on reports and such, make the result field in the TIME opcode 12 bytes long. The TIME operation within RPG will return the system date as opposed to the session date if the first six bytes contain the time and the last six bytes contain the system date.

Another solution to your problem is simply to sign off the console CRT each day. Alternatively, you could write a program to retrieve the system date and place it in the LDA. Place the statement // DATE ?L'1,6'? after the // RUN statement for the program that retrieved the date (assuming the date was stored beginning in LDA position 1).

Necessity of IPLs

answered by Mel Beckman

Q I read with interest a Technical Corner answer that described how to change the system date on a S/36 without performing an IPL. I would like to know whether there is a similar solution for changing the system time. We are running an on-line system that must be available 24 hours a day, seven days a week, so we do not IPL every day. It would be nice not to take the system down for an IPL when we change to daylight savings time in the spring or to standard time in the fall.

A Changing the system time requires an assembler subroutine that converts the time of day to system timer units elapsed since midnight. (System timer units are expressed in binary as multiples of 8.192 milliseconds; few people can do the conversion in their heads.) If you don't have the BAL assembler (and most people don't), you're out of luck.

Worse, if you did succeed in changing the system time without an IPL, you would cause the system to do all sorts of abnormal things. The abnormalities occur because several system functions rely on timer queue elements (TQEs) to delay their execution for an appropriate interval, after which they "wake up," perform their functions, and "go back to sleep." The ERAP routine, for example, wakes up every six minutes to post various system event counters, and the midnight date change routine wakes up at system midnight to change the system date. These routines wait for a specific value of the system time counter to trigger their execution, so if you advance the system time beyond the expected value, the execution of these routines may be delayed up to 24 hours longer than desired. The results are unpredictable and messy. Your jobs may (choose one): (1) be executed normally, (2) "hang" indefinitely, or (3) die a miserable death.

Your question brings up another issue as well. Apparently, you are performing an IPL only twice a year, which is not nearly often enough; once a month would be better. During IPL, the S/36 performs a number of diagnostic tests that ensure the reliability of system operation. It also applies Program Temporary Fixes (PTFs) that you may have loaded to correct SSP bugs or improve system performance, and it reorganizes file indexes (which also improves performance). Regular IPLs are absolutely necessary to keep your S/36 healthy.

Running PTF Procedure LDMARES

answered by Jeffrey Pisarczyk

Q Soon after I installed PTF 3700 on my S/36, a colleague asked me whether I ran procedure LDMARES while applying the PTF. I have never heard of procedure LDMARES — what is it? Am I going to encounter problems because I didn't use it?

A In PTF levels 3700 and up, the SSP's $MARES module is loaded as part of the PTF process. If this module can't find enough room for itself in #LIBRARY, it overwrites anything in its way, which means you may lose part of your SSP. Including procedure LDMARES in your PTF installation stops $MARES from overwriting your #LIBRARY. The new steps for PTF levels 3700 and above are:

- Do a PTF COPY,ALL of your 3X00 PTFs
- Run procedure LDMARES by keying in LDMARES and letting the procedure do the work
- IPL the system
- Run PTF APPLY

These steps are outlined in the PTFNEWS library supplied with your PTF diskettes.

You may or may not encounter problems because you didn't use procedure LDMARES. Not using it could cause task dumps with SRC-0090 codes, and messages such as "SYS-2599: IBM load module has invalid table" and "SYS-3820: Invalid data found in procedure being processed." Beyond these indications, because $MARES writes over anything in its way, the symptom list could contain just about any problem. In short, if your system exhibits abnormal behavior, look at $MARES first.

To determine whether $MARES is your problem, run a PTF LIST,,CRT. If you find four or more consecutive asterisks in the PTF log, you have some kind of PTF-based problem brewing.

To correct the problem, you must either reload your SSP from a reliable backup developed before you installed the PTF, or you can reinstall

Release 5.1 of your SSP from the PID diskettes. Then install the PTF and run procedure LDMARES. If you don't experience any SSP problems, you should be able to bypass the reload and simply reinstall the PTF(s).

Upgrading to a New S/36

answered by Rick Graham, Gary T. Kratzer, and Dan Stephens

Question: We plan to upgrade from a S/36 5362 C02 to a 5360 D25 and have already received permission from IBM to transfer our software licenses. What is the best way to actually move our code? For example, we have some customized members in #LIBRARY. Could we actually back up #LIBRARY from the C02, IPL the new system from diskette, and load #LIBRARY on the new system? Then, could we back up all the other libraries (including both IBM's and our own) and restore them? Or should we just bring up the whole system from scratch?

A First, build an empty save library on your 5362, and then do a LIBRLIBR from #LIBRARY to copy all user members and changed IBM members to the save library. Back up your entire 5362. On your new system, do a complete rebuild/reload using the PID diskettes; then restore all user libraries, files, and so on. Next, LIBRLIBR the user members from the save library back to #LIBRARY (or use POP to copy them back over so you don't have to dedicate the system). Finally, apply PTFs.

Although the IPL-from-diskette scenario you mentioned would probably work okay, we don't think it would save you much time, and there may be danger in bypassing the CNFIGSSP reload procedure. Why risk doing it wrong and then having to do it right afterward, instead of doing it right the first time?

All in all, it is better in the long run to bring up the new system from scratch. You can do it while you continue to run on your present system, and you won't have to "pull the plug" until you are sure that everything is all right.

Tapes

CHAPTER

19

Deciphering the Tape Header Label Format

answered by Matthew Henry

Q I am trying to find the layout of the user labels on a tape created by a SAVE operation to allow data interchange from another system. IBM's *System Data Areas* (LY21-0592) says the labels contain portions of the EMBEDDED F1. I'm at a loss when it comes to finding a detailed breakdown of the header label. Any suggestions?

A One solution to your question might be the S/36 command called TAPECOPY, which translates S/36 files into a generic format on tape in either ASCII or EBCDIC. Another place to look for the answer to your question is the COPYT macro under "Tape User Labels Layout" in *System Data Areas*. Program $COPY (run by SAVE) stores the entire user format-1 (VTOC entry) in two 80-byte parts and is detailed in "User Format-1" in System Data Areas.

Tape label records are always 80 bytes long and are commonly referred to as headers. The embedded format-1, which is 128 bytes long, is divided between two tape header label records. The first 68 bytes are placed in the first tape label (HDR1) starting at offset X'C', and the remaining 60 bytes are placed in the second tape label (HDR2) starting at offset X'5'. Information in *System Data Areas* shows how the $COPY program assembles the information.

Reading Tapes with Nonstandard or Missing Labels

answered by Mel Beckman

Q Is it possible for the S/36 (SSP Release 5.1) to read unlabeled tapes or tapes using nonstandard labels on the 8809 tape drive?

A You can read nonlabeled tapes on the S/36 using the TAPECOPY procedure, specifying NONLABEL for parameter number 9. With nonlabeled tapes, though, you must write a separate program to extract logical records from the physical tape blocks — TAPECOPY creates a file with one record per tape block.

You also can use nonstandard labels on the S/36 to a limited extent (i.e., nonstandard labels are ignored). On a tape with nonstandard labels, the system reads only the first file on the tape (i.e., from the first tape mark to the second tape mark). With the S/36 TAPECOPY features, though, you can tell the system to treat tapes with nonstandard labels as nonlabeled tapes — by specifying either BYPASS or BLP in parameter 9. By bypassing label processing on tapes with nonstandard labels, you have access to the data stored in all files on the tape.

Preventing Tape Rewind When Saving Individual Items

by Alex Barish

Code on diskette:
Procedure LOOPSAVE

On the S/36, when you individually copy several items to tape, the tape rewinds at the end of each command (e.g., at the end of a FROMLIBR, SAVE, or TAPECOPY command), despite the use of the LEAVE parameter. For normal backups, the rewind presents no problem because all the necessary commands can be coded within the same procedure, and the tape will not rewind until the initial procedure is completed.

To eliminate the rewind problem when you are saving individual items not included in your normal backup routine, I offer procedure LOOPSAVE (Figure 19-1). Procedure LOOPSAVE encloses in a loop a HELP OCL statement that prompts you for the desired SSP backup command (e.g., SAVE, FROMLIBR, TAPECOPY). Because procedure LOOPSAVE does not end until you enter END (instead of HELP) as the first parameter, the tape does not rewind after each command.

Figure 19-1 *Procedure LOOPSAVE*

```
// * ' THIS PROCEDURE ALLOWS THE LEAVE PARAMETER TO BE USED FOR A TAPE'
// * ' JOB (MULTIPLE TAPE JOBS WITHIN ONE JOB NUMBER) '
// * ' NORMALLY THE TAPE IS REWOUND FOR EACH NEW JOB  THIS PROCEDURE'
// * ' ALLOWS YOU TO RUN A FEW JOBS (JOB STEPS) WITHIN THIS PROCEDURE '
// * ' EFFECTIVELY. THE SYSTEM WILL TREAT THIS PROCEDURE AS A SINGLE'
// * ' JOB. HENCE ALLOWING THE LEAVE PARAMETER '
// TAG AGAIN
// IF ?1R?-END RETURN
// IF ?1R?-HELP GOTO HELP
?1R?
// EVALUATE P1-''
// GOTO AGAIN
*
// TAG HELP
// HELP ?2R?
// EVALUATE P1-''
// EVALUATE P2-''
// GOTO AGAIN
```

IPLing from Tape

answered by Matt Drage

Q Given the ability of the S/36 (5360) to back up #LIBRARY to tape, will the system perform an initial program load (IPL) from tape when a MODE SELECT: F (MSIPL from diskette) is keyed at the system control panel? If so, how does the system know that the IPL should be from tape and not diskette?

A If the S/36 has a tape drive configured, the tape drive is the primary external IPL device. When you perform a "reload" MSIPL, the reload will be from tape if the Control Storage Processor (CSP) finds a tape loaded and ready in the tape drive. (On a 5360, initiate a MSIPL reload with "Mode Select F"; on a 5362, use "Function Select 2" instead.) If the CSP does not find a tape, the reload will be from diskette. You can use either the 8809 reel-to-reel drive or the 6157 cartridge drive for an external IPL. You must use the SAVELIBR #LIBRARY procedure to create the tape from which you will reload.

Workstations

CHAPTER

20

Retrieving Cursor Position in Demand or Primary Workstation Files

answered by Mike Patton

Code on diskette:
Assembler subroutine SUBRCP

Q In a previous BitStop, you published a way to retrieve the screen cursor position in an RPG program via an assembler subroutine. The routine accessed the workstation as a demand file. Is there a way to retrieve the cursor position when accessing the workstation as a primary file?

A The assembly language subroutine SUBRCP retrieves the position of the screen cursor during the most recent workstation operation. It should be coded as follows:

```
C                          EXIT SUBRCP
C                          RLABL          ROW      30
C                          RLABL          COLUMN   30
```

The routine works with demand files or primary files on the S/36. Note that the RPG EXIT operation that calls the routine need not occur immediately following a READ operation when the workstation file is a demand file. You can condition the EXIT operation however you wish.

Re-creating Subroutine SUBRCP

If you don't have assembler subroutine SUBRCP, you can re-create it with procedure MKSUBRCP (you don't need IBM's Assembler Language Program Product to install SUBRCP). You must have first compiled program MAKMEM (see *Transmitting S/36 Object Code,* page 38) to run MKSUBRCP. You need to run MKSUBRCP only once to create the SUBRCP subroutine.

```
// * 'Re-creating R-module SUBRCP in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000071'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-+
'D8E2E4C2D9C3D7404000000040000000000000B000000000990002200000000888'
// LOCAL OFFSET-273,DATA-+
'0B2B17383100000000000000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-26,EXTEND-26
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBRCP member to insert object code
// LOAD $FEFIX
```

Continued

// RUN
HDR 3848 SUBR[illegible] RPGLIB
PTF 138A RSUBRCP [illegible]
C800 00 0000 E20BE2[illegible]
00 0020 0000000000[illegible]
00 0040 E3920032340800[illegible]
[illegible]

```
// RUN
HDR  384B SUBRC00000
PTF  136A RSUBRCP,99,,#RPGLIB
DATA C9D0 00 0000 E20BE2E4C2D9C3D700000000A500000000000000000000000000000000000000
DATA 5596 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 6A09 00 0040 E3320032340800523401004A3402004E35010052370100A27501007501037501
DATA 85A7 00 0060 05D20136350200 52B50202370200A0C0870053D20101350 02E2A23130F080703
DATA 75C2 00 0080 E3300063020052B50205370200A0C08700630E01005200A4C2010000C2020000
DATA 4175 00 00A0 C0870000340800981C00009C000C00009D009C387700002E2C272313110D0902
DATA D2DC 00 00C0 E3310095009D0802009E009D3B88009CA7010101880202009C861002009EA601
DATA 256F 00 00E0 0202A6010202A6110202A6020202B62002009C8620020000002E191408070501
DATA 0CDC 00 0100 E305009B009DC087000000000000000000000000000000000000000000000000
DATA 94A8 00 0120 0000000000000000000000000000000000000000000000000000000000000001
DATA E2FB 00 0140 E30500A40002000500060000000000000000000000000000000000000000000000
DATA 8BC6 00 0160 0000000000000000000000000000000000000000000000000000000000000000
DATA 1387 00 0180 C5FFFFA400000000000000000000000000000000000000000000000000000000
DATA 124F 00 01A0 0000000000000000000000000000000000000000000000000000000000000000
DATA EDA6 00 01C0 615C000000000000000000000000000000000000000000000000000000000000
DATA 0D6E 00 01E0 0000000000000000000000000000000000000000000000000000000000000000
END  8EC5
```

Reading Screen When Roll Key Pressed

answered by Gary T. Kratzer

Code on diskette:
Assembler subroutine SUBRRR

Q On our S/36, we use programs with WORKSTN (workstation) files. When the Roll keys are pressed, RPG bypasses the move field logic and no fields are changed. I would like to know if there is a way to get the system to accept data changes when function keys are pressed. Note: IBM's POP works that way when you edit members.

A I, too, faced the annoying limitation of the data from input-capable fields not being returned to the program when a Roll key was pressed. However, I discovered that the hardware actually does return input data to the program buffer. But the S/36 RPG object program fails to move the buffer into the input record fields. Whether this is by design or oversight is academic. A very simple assembler language subroutine can make things right.

To use the subroutine, code an EXIT SUBRRR statement immediately after each workstation READ operation for which you want Roll key data returned:

```
C                   READ WORKSTN               1111
C                   EXIT SUBRRR
```

It is important that you don't condition either the READ or EXIT operation with any indicators in positions 9 through 17 and that no statements come between the READ and EXIT statements. After a READ operation, SUBRRR will check to see whether any function keys were pressed. If they were, the workstation field move routine, which is already part of your program, is called to move data from the workstation input buffer into your fields.

Re-creating Subroutine SUBRRR

If you don't have assembler routine SUBRRR, you can re-create it with procedure MKSUBRRR (you don't need IBM's Assembler Language Program Product to create SUBRRR). To run MKSUBRRR, you must be signed on as a security officer.

```
**
** Use $MAINT to copy SUBR20 to SUBRRR so we have a subroutine member to patch
**
// LOAD $MAINT
// RUN
// COPY FROM-#RPGLIB,TO-#RPGLIB,LIBRARY-R,NAME-SUBR20,NEWNAME-SUBRRR,RETAIN-R
// END
**
** Patch the new SUBRRR member to make it return roll key data
**
// LOAD $FEFIX
// RUN
HDR
PTF RSUBRRR,,#RPGLIB
DATA 7C,0002,E2E4,C2D9,D9D9
DATA 38,0041,30,0030,3408,0048,3408,004A,3401,0040,3402,0044,0F01,004A
DATA B4,0068,004C,3602,004A,B502,00B5,0203,8804,00F2,9017,0F01,004A,004E
DATA 3D,0070,3501,004A,1C00,2F2B,2919,1613,0F0B,0703
DATA 2E,0081,1D,004E,0100,3C00,C201,0000,C087,0000,C201,0000,C202,0000
DATA 06,0098,C087,0000,0000,0005,0007
DATA 00,00BE,0002
END
```

Enabling Function and Command Keys Dynamically

by Mel Beckman

Code on diskette:
Assembler subroutine SUBREK

Often in a S/36 interactive RPG program, you need to restrict the set of command and function keys available to the user. The usual way to do this is by compiling, in advance, separate screen format members, each with a unique combination of command and function keys enabled on the $SFGR S-spec. But because you don't always know which keys you want to be valid until your application is running, you need a way to enable command and function keys dynamically.

Assembly language subroutine SUBREK provides this ability. You specify which command and function keys are to be enabled by passing a four-byte "bit mask" field to SUBREK in an RLABL statement ('A' in Figure 20-1). Use the RPG BITON and BITOF operations to set the appropriate bits in the four-byte field using a data structure ('B' in Figure 20-1) to redefine each byte.

The meanings of each bit are shown in Figure 20-2.

Figure 20-1

Example of using SUBREK

```
*        1         2         3         4         5         6         7         8
'B'  I            DS
     I                                          1   4 MASK
     I                                          1   1 MASK1
     I                                          2   2 MASK2
     I                                          3   3 MASK3
     I                                          4   4 MASK4

'A'  C                         BITOF'01234567'MASK1              Clear all bits
     C                         BITOF'01234567'MASK2
     C                         BITOF'01234567'MASK3
     C                         BITOF'01234567'MASK4
     C*
     C                         BITON'046'     MASK1              Set ckeys 1, 5, 7
     C                         BITON'7  '     MASK3              Set ckey 24
     C                         EXIT SUBREK                       Enable keys
     C                         RLABL          MASK
```

Figure 20-2

Bit values for field MASK

```
Byte 1, bit 0  enable command key 1
        bit 1  enable command key 2
        bit 2  enable command key 3
        bit 3  enable command key 4
        bit 4  enable command key 5
        bit 5  enable command key 6
        bit 6  enable command key 7
        bit 7  enable command key 8
Byte 2, bit 0  enable command key 9
        bit 1  enable command key 10
        bit 2  enable command key 11
        bit 3  enable command key 12
        bit 4  enable command key 13
        bit 5  enable command key 14
        bit 6  enable command key 15
        bit 7  enable command key 16
Byte 3, bit 0  enable command key 17
        bit 1  enable command key 18
        bit 2  enable command key 19
        bit 3  enable command key 20
        bit 4  enable command key 21
        bit 5  enable command key 22
        bit 6  enable command key 23
        bit 7  enable command key 24
Byte 4, bit 0  pass back print key
        bit 1  pass back roll-up key
        bit 2  pass back roll down key
        bit 3  pass back clear key
        bit 4  pass back help key
        bit 5  pass back record backspace key
        bit 6  unused
        bit 7  unused
```

Re-creating Subroutine SUBREK

If you don't have assembler subroutine SUBREK, you can re-create it with procedure MKSUBREK (you don't need IBM's Assembler Language Program Product to install SUBREK). You must have first compiled program MAKMEM(see *Transmitting S/36 Object Code,* page 38) to run MKSUBREK. You need to run MKSUBREK only once to create the SUBREK subroutine.

```
// * 'Re-creating R-module SUBREK in library #RPGLIB '
* Build an empty member in a $MAINT file with the correct directory entry
// LOCAL OFFSET-201,DATA-'00000039'    Number of $MAINT records
// LOCAL OFFSET-209,DATA-'
```

Continued

```
'D9E2E4C2D9C5D240400000002000000000000060000000000990001200000001687'
// LOCAL OFFSET-273,DATA-+
'1124135631000000000000000000000000000000000'
// LOAD MAKMEM
// FILE NAME-BINARY,LABEL-$MAINT,RETAIN-J,BLOCKS-25,EXTEND-25
// RUN
* Copy renamed member to target library
// LOAD $MAINT
// FILE NAME-$MAINT,RETAIN-S
// RUN
// COPY FROM-DISK,FILE-$MAINT,RETAIN-R,TO-#RPGLIB
// END
* Patch the new SUBREK member to insert object code
// LOAD $FEFIX
// RUN
HDR  3854 SUBRE00000
PTF  2323 RSUBREK,99,,#RPGLIB
DATA 697B 00 0000 E20BE2E4C2D9C5D200055A00550000000000000000000000000000000000000
DATA F23E 00 0020 0000000000000000000000000000000000000000000000000000000000000000
DATA 7735 00 0040 E334058E340806A8340105A0340205A435020099B50205BDC000F2810CBDFF08
DATA 5CE7 00 0060 F2811E850209F18712350105A8750102BA60389C0044001C0300000028080703
DATA 82EF 00 0080 E31B05AA05AE008C023B05AD0E0105A805AAC2010000C2020000C08700000003
DATA 82E9 00 00A0 000000000000000000000000000000000000000000000000000000000D0B0701
DATA 63DA 00 00C0 C5FFFFAA00000000000000000000000000000000000000000000000000000000
DATA 83A1 00 00E0 0000000000000000000000000000000000000000000000000000000000000000
END  35F8
```

Reading Under Format

by Perry Gardai

Code on diskette:
Procedures LONGPROC, PROMPT1, PROMPT, PROMRUF

While browsing through S/36 manuals, you may have run across a few vague references to an RPG II programming technique called RUF (Read Under Format), which appears to be of some benefit to interactive programs. Being the curious creatures we data processing people are, such references spark our imaginations; however, trying to track down any consolidated information on RUF is difficult. The manuals give us a sentence here and a paragraph there but never really tell us under what conditions RUF might be used. After doing some research on the subject, I have discovered that by using RUF, you can not only get tough with RPG II programs that are slow to initiate but also transfer prompt-screen data between two or more related programs.

RUF essentially allows a screen to be displayed and used before the on-line program that processes the data from the screen becomes resident in memory. As soon as the screen is displayed, users can enter data or use any other function the screen allows. They do not have to wait for the system to go through housekeeping routines associated with loading the program that processes the screen. The RUF technique can be implemented within a procedure by using the // PROMPT statement or within an on-line RPG program by using external switches. When using RUF with // PROMPT, you can save some operator time if, in the procedure that loads the on-line program, numerous OCL statements precede the actual program call or if a program is slow to initiate. When using RUF within an on-line RPG program, you can output a screen, exit the program, return control to the calling procedure, and load another program to process the screen, all while the operator continues to enter data on the screen. This second implementation of RUF is known as "program switching."

This article will examine both uses of RUF. I assume that the reader is familiar with the use of the // PROMPT statement, SDA, and on-line programming techniques.

Passing Prompt-Screen Data to an On-line Program

When associated with the // PROMPT statement, the RUF technique normally is used to display the first screen processed by an on-line program. To specify that input from the screen is to be used as program data, the programmer codes the PDATA-YES parameter of the // PROMPT statement. Usually, the // PROMPT statement is separated from the on-line program's // LOAD statement by several intervening OCL statements.

To illustrate, assume that a procedure, evoked from a menu option, contains many OCL statements before it loads an on-line program that processes four screens, as in the case of example procedure LONGPROC (Figure 20-3). Further assume that the first screen used by the program is a typical front-end screen that prompts for an account number or name and an action code (e.g., add, change, or delete). By displaying this screen (e.g., SCREEN1 of the // PROMPT statement in Figure 20-3) before the program is called by the procedure, the operator can make preliminary selections while the system processes the OCL between the // PROMPT and the // LOAD statement.

When the program is loaded, the system knows that a screen has been displayed that will contain data to be passed to the on-line program (i.e., the PDATA-YES parameter has been specified to the // PROMPT statement), so the system waits for the user to press Enter before it begins processing the program. Although the system may not have processed the procedure's // LOAD statement by the time the Enter key is pressed, the time elapsed between when the user chose the menu option associated with procedure LONGPROC and when the system displayed SCREEN1 is minimal.

To further illustrate how RUF works with // PROMPT, let's examine two procedures — one without RUF (Figure 20-4) and one with RUF (Figure 20-5) — that attempt to accomplish the same goal. Each example uses a // PROMPT statement to display a screen to which the operator inputs a range of accounts to be listed by an on-line program. Assume that a prompt screen has been developed that prompts the operator for the first and last account numbers to be listed.

On the S/36, the traditional approach to locate records within the specified range is to pass the entries from the prompt screen to a selective sort in the form of substitutional parameters (Figure 20-4). The sort then extracts the needed records from the input file (AP.APPRO) and sends then to an output file (APAPPROS). The output file is then passed to a simple print program (PROMP1) that lists the accounts.

The alternative to this method is the use of the RUF technique (Figure 20-5). As in the traditional method, the prompt screen is displayed with all of the default values preloaded into the appropriate parameters via // EVALU ATE statements. But unlike the traditional approach, the PDATA-YES parameter is used with the // PROMPT statement so the prompt screen data is passed to the on-line program (PROMPT) instead of to substitutional parameters within the procedure. The program then edits the data input from the screen (via a workstation file) and uses sequential-within-limits processing to extract the desired records from file AP.APPRO. Again, as soon as the prompt screen is displayed, the operator can enter the required data, even though the // LOAD statement for on-line program PROMPT has not been executed.

One of the major advantages to the technique demonstrated in Figure 20-5 is that the on-line program that processes the prompt screen can edit the screen values much more extensively than is possible with OCL statements. For instance, from within on-line program PROMPT (i.e., the program loaded by the procedure in Figure 20-5), account numbers can be validated against the account master file via a chain operation. If edit errors are detected, program PROMPT can redisplay the prompt screen and, by using standard field attributes such as reverse image and cursor positioning, draw the operator's attention to the errors. On this point, Figure 20-5's approach is certainly preferable if the data being entered by the operator is at all operationally sensitive.

A second advantage is performance. Because the technique shown in Figure 20-5 eliminates the need for a selective sort, it is more efficient than the technique shown in Figure 20-4; the selective sort in Figure 20-4 would have to process the entire file before the print program is executed. By using limits processing in the on-line program and // PROMPT's PDATA-YES parameter in the calling procedure, the technique in Figure 20-5 processes only these records that fall within the specified range of account numbers — not the entire file. If you are using a large file and need to process only a few records, the RUF technique combined with sequential-within-limits processing can dramatically improve throughput.

When applying the RUF technique via the // PROMPT statement to an on-line program, you should note that execution of the program varies somewhat from that of a standard on-line program. When the // LOAD statement for a standard on-line program is executed, SSP determines the absence of an existing screen format member and passes a blank screen to the program. The programmer, in turn, must allow for the program to process this blank screen by testing for a "catch-all" screen indicator (unconditional indicator attached to the Workstation Input Specification) during the first processing cycle. When the program reaches output time during the first cycle, the first program screen — conditioned by an indicator set on when the blank screen was tested — is displayed.

When a // LOAD statement for a program is executed after RUF is implemented, Workstation Data Management determines that a screen format is already active. Therefore, the blank screen is not passed to the program. Rather, the first processing cycle is suspended until the operator presses Enter or in some other manner transmits input to the program (e.g., use of command or function keys). Between the time the prompt screen is initially displayed and the time the operator transmits input, the system continues to execute the OCL statements, load the on-line program, and perform all housekeeping functions required to run the program. When the operator presses Enter or otherwise transmits input, the first processing cycle is executed. From this point on, the program works like any other on-line program.

Does calling a screen from a procedure and then loading an on-line program (i.e., using the RUF technique) have any functional advantage over first loading the on-line program and then outputting the screen? Frankly, the answer depends on your application code and the complexity of your procedures. Because the on-line program associated with RUF works like any other on-line program once the operator presses Enter for the first time, all of the inherent advantages of an on-line program are in place. The advantage you gain by calling the screen from the procedure instead of the on-line program is the ability to display the screen format before the on-line program is loaded. In many applications, the RUF technique may eliminate lag time between when your users select a menu option and when they can begin to enter data.

Program Switching

The second major use of the RUF technique allows for program switching, the passing of data between two or more interactive programs from within the same procedure. Program switching is accomplished in part by displaying a screen format from within an active on-line program that subsequently will be used as the first input screen to another program. Such a technique might be necessary to circumvent the 64 K program size limitation imposed by SSP or to allow a smooth transition between two related programs, such as a master file maintenance program and its related inquiry program.

To illustrate, assume you have written a 62 K on-line interactive program that uses 10 screens and 13 files. A month or two later, because of user requirements, an additional screen and file must be added to the already large program. Now, because you included the extra screen and file, the program won't compile in 64 K — even with overlays.

What are the alternatives? You could tell the users that it can't be done. You could do some bit counting of the operation codes to make the program more efficient. You could remove all the editing you have programmed in so carefully. Or you can split the program. The only acceptable alternative is to split the program into two logical parts that operate as one.

Nearly every program has some point that logically separates one set of screens from the rest. For example, a program that maintains both header and line item functions could be divided logically between the two function types. The trick is to split the program in such a way that the users are not aware they are using two programs instead of one. Therein lies the beauty of the RUF technique.

Assume that the 62 K program mentioned earlier has been divided into two unique programs — PROGRAMA and PROGRAMB — and a procedure, PROMRUF, which contains both programs calls in the proper sequence of execution (Figure 20-6). That is, PROGRAMA is always the entry point for the two programs. It contains the bulk of the code and may well be a stand-alone program. PROGRAMB, on the other hand, is the program to which PROGRAMA always switches when it reaches a certain point or requires a certain function be performed. PROGRAMB contains the screens that are logically separate but still need to be accessible by PROGRAMA. The structure of the procedure allows both programs to be executed in a circular fashion — that is, exit PROGRAMA, enter PROGRAMB, and vice versa.

To let the procedure know which program in the loop is to be activated, each program sets on an external switch at the time it is exited with the intent of entering the other program. If the user requests a normal EOJ (End of Job) exit, the switch is not set and the procedure ends. If control is to be transferred to the second program, the switch is set on, the program is exited, and control returns to the procedure, which evaluates the switch and activates the second program.

For the transition from one program to the other to be manageable, each program must "know" what to expect as input during its first processing cycle. The easiest way to establish this control is to use the same screen to enter PROGRAMB that you use to exit PROGRAMA and to use the same screen to exit PROGRAMB that you use to re-enter PROGRAMA. Granted, this methodology causes some redundant screen format members and, at times, redundant maintenance, but from a programmer's point of view, it allows for easy control of the transfer of data between programs. You can ensure that your duplicate screen formats are identical in every way by using the include function of SEU (Command key 11) while in SDA.

Visually, the program structures may look like the structure in Figure 20-7. In this example, after Enter has been pressed on screen PRAS4 in PROGRAMA, an UPSI switch is set, screen PRBS1 is displayed, and PROGRAMA goes to end of job. Control of processing is then returned to the procedure, which reads the switch settings and determines that PROGRAMB is to be activated. While this determination is being made, the user is still able to enter data into screen PRBS1 just as if PROGRAMA were still active. By the time the user presses Enter on screen PRBS1, PROGRAMB is active and reads the screen as input for its first full processing cycle. This fact implies that the SSP does not pass a blank screen to PROGRAMB, nor is there a blank screen indicator in the I-specs for PROGRAMB.

The return from PROGRAMB to PROGRAMA is identical in nature

(Figure 20-7). When Enter is pressed on screen PRBS4, a different switch is set (in this case, switch 2), screen PRA55 is displayed, and PROGRAMB goes to EOJ. Again, control is passed to the procedure while the operator is entering data on screen PRA55. The procedure evaluates the current UPSI switch setting and determines that PROGRAMA is to be reactivated. When the user presses Enter, the screen ID for screen PRSA5 is read into the program, the corresponding input indicator is set on, and processing continues as if PROGRAMA had always been active.

This technique can also be used to transfer back and forth between two related standalone programs, such as a file maintenance program and an inquiry program. By using the LDA to pass information about the key to the next program, either program can be entered at a point other than the initial record selection screen. As an example, assume both the payroll master file maintenance program and the inquiry program have an initial entry screen that asks for an employee's social security number as the key to the file.

Regardless of which program the user initially enters, the key for the subsequent work has been established. At some predetermined juncture in each program, transfer to the companion program can be requested (normally, by a command key), at which time the current record's key is loaded into the LDA, and the program goes to EOJ. Because the key to the current data record is already known, there is no need to redisplay the initial entry screen associated with either program. Rather, by reading the screen ID from the prior program, reading in the key value stored in the LDA, and chaining to the appropriate data record, a detailed data screen can be displayed when the transfer is requested. This implementation of the RUF technique saves users from having to exit the first program and then take another menu option to enter the second one.

One last point needs to be made about the RUF technique. When you must pass a great deal of data between two programs, you can use the RUF technique in conjunction with one of two other methods. The first and easiest is to use the LDA. If the programs are SRT (Single Requester Terminal) programs, the LDA will be read automatically during the first input cycle and written out at EOJ. If the programs are MRT (Multiple Requester Terminal) programs, IBM external subroutines SUB20 and SUB21 must be used to read the LDA and set and read the external UPSI switches. If more data must be passed between the programs than will fit in the LDA, a second method is to output the data to a file the key to which is the workstation ID of the workstation being released from the program. On the first cycle of the program being entered, simply use the workstation ID, captured by the KWSID continuation line of the workstation F-spec, and chain to the record to retrieve the data.

So, are these the types of applications IBM envisioned for RUF? Who knows? What I do know is that these techniques do work, do save time, and can be the answer to some rather sticky technical problems. So, don't let performance BYTE you. Just RUF back.

Figure 20-3

Sample procedure LONGPROC

```
*****************************************************************************
*  LONGPROC   THIS IS A SYSTEM/36 PROCEDURE THAT USES THE READ UNDER FORMAT    *
*             'RUF' TECHNIQUE FOR PASSING DATA FROM A PROMPT SCREEN TO A       *
*             MULTI-SCREEN WORKSTATION PROGRAM                                 *
*   1   SET DEFAULTS IN THE SCREEN HEADING BEFORE PROMPT SCREEN IS DISPLAYED   *
*   2   DISPLAY PROMPT SCREEN WITH DEFAULTS                                    *
*   3   BUILD ALL NECESSARY FILES AND DO OTHER REQUIRED OCL                    *
*   4   LOAD AND RUN THE WORKSTATION PROGRAM.                                  *
*****************************************************************************
*
// EVALUATE P1-'1'                              SET SCREEN INPUT/OUTPUT ID
// IF ?TIME?>120000 EVALUATE P2-?TIME?PM        SET TIME TO AM OR PM
//   ELSE EVALUATE P2-?TIME?AM
// EVALUATE P3-'?USER?' P4-'?WS?'               SET HEADING DEFAULTS
// EVALUATE P5-01,2 P6-' 1' P7-'    1'          SET FROM ACCOUNT DEFAULTS
// EVALUATE P8-99 P9-99 P10-9999                SET TO ACCOUNT DEFAULTS
*
// PROMPT MEMBER-LONGPRFM,FORMAT-SCREEN1,LENGTH-'1,8,,2,2,2,4,2,2,4,40',+
   PDATA-YES                                    DISPLAY PROMPT AND PASS DATA TO PROGRAM
*
NOHALT 3,JOB                                    THE OCL FROM THIS POINT ON IS
// ALOCATE UNIT-I1                              SIMPLY FOR THE SAKE OF
// IF DATAF1-TRANSACT SAVE TRANSACT             DEMONSTRATION, TO ILLUSTRATE
// DEALLOC UNIT-I1                              A SIGNIFICANT TIME LAG BETWEEN
*                                               THE TIME THE PROMPT SCREEN IS
// IF DATAF1-AUDLST?WS? DELETE AUDLST?WS?,F1    EXECUTED AND THE TIME THE
// IF DATAF1-TRANSACT  DELETE TRAANSACT,F1      PROGRAM THAT PROCESSES THE
BLDFILE AUDLST?WS?,S,RECORDS,1000,128           PROMPT SCREEN IS ACTUALLY
BLDFILE TRANSACT,,RECORDS,1000                  LOADED INTO MEMORY
// IFF DATAF1-ALPINDX BLDINDEX ALPINDX,12,24,APMASTER
*
// ATTR CANCEL-NO,INQUIRY-NO,MRTMAX-05,PRIORITY-MEDIUM
*
// LOAD LONGPR                                  LOAD MULT-SCREEN ON-LINE
// FILE NAME-APAPPRO,LABEL-AP.APPRO,DISP-SHR    PROGRAM TO PROCESS PROMPT
// FILE NAME-APMASTER,DISP-SHR                  SCREEN AND CONTINUE ON WITH
// FILE NAME-AUDLST,LABEL-AUDLST?WS?,DISP-SHR   REST OF NORMAL ON-LINE
// FILE NAME-TRANSACT,DISP-SHR                  PROCESSING SEQUENCE
// RUN
```

Figure 20-4

Sample procedure PROMPT1

```
*****************************************************************************
*  PROMPT1    THIS IS A SYSTEM/36 PROCEDURE THAT USES A PROMPT SCREEN TO PASS  *
*             DATA TO A SORT, WHICH IN TURN LOADS A PROGRAM TO PRINT A         *
*             REPORT FROM THE SORTED DATA                                      *
*   1   SET DEFAULTS IN THE SCREEN HEADING BEFORE PROMPT SCREEN IS DISPLAYED   *
*   2   DISPLAY PROMPT SCREEN WITH DEFAULTS AND TEST FOR CMD/7                 *
*   3   EDIT THE PROMPT SCREEN ENTRIES   IF THE EDITS FAIL, SET SWITCHES TO    *
*       HIGHLIGHT AND POSITION CURSOR AND REDISPLAY PROMPT                     *
*   4   IF EDITS ARE GOOD, THEN CALL THE SORT TO SELECT THE RANGE OF ACCOUNTS  *
*   5   RUN THE PROGRAM TO PRINT THE REPORT                                    *
*****************************************************************************
*
// EVALUATE P1-'1'                              SET SCREEN INPUT/OUTPUT ID
// IF ?TIME?>120000 EVALUATE P2-?TIME?PM        SET TIME TO AM/PM
//   ELSE EVALUATE P2-?TIME?AM
// EVALUATE P3-'?USER?' P4-'?WS?'               SET HEADING DEFAULTS
// EVALUATE P5-01,2 P6-' 1' P7-'    1'          SET FROM ACCOUNT DEFAULTS
// EVALUATE P8-99 P9-99 P10-9999                SET TO ACCOUNT DEFAULTS
*
// TAG AGAIN                                    REDISPLAY PROMPT ON ERROR
*
// PROMPT MEMBER-PROMPTPM,FORMAT-SCREEN1,LENGTH-'1,8,,2,2,2,4,2,2,4,40',+
   UPSI-YES                                     OUTPUT PROMPT SCREEN
// IF ?CD?/2007 CANCEL                          IF CMD/7 CANCEL THIS JOB
*
// SWITCH 00000000                              SET ALL ERROR CONTROLS OFF
// IF ?5?/ SWITCH 10000000                      VALIDATE EACH ENTRY TO INSURE
// IF ?6?/ SWITCH X1000000                      IT IS NOT BLANK   IF AN ENTRY
// IF ?7?/ SWITCH XX100000                      IS BLANK, SET RELATED SWITCH
// IF ?8?/ SWITCH XXX10000                      TO POSITION CURSOR AND HIGHLIGHT
// IF ?9?/ SWITCH XXXX1000                      THE FIELD
// IF ?10?/ SWITCH XXXXX100
*
```

```
// IF SWITCH-00000000 GOTO OK                    IFF ANY ERRORS THEN PROCEDE
//   ELSE EVALUATE P11-'1 OR MORE ERRORS, CORRECT AND REENTER'
// GOTO AGAIN
*
// TAG OK
*
// LOAD #GSORT                                   EXECUTE SORT TO INCLUDE ONLY
// FILE NAME-INPUT,LABEL-AP.APPRO                THE RANGE OF ACCOUNTS REQUESTED
// FILE NAME-OUTPUT,LABEL-APAPPROS,RECORDS-?F'A,AP.APPRO'?,RETAIN-J
// RUN
   HSORTA     8A        3        N
   I C   2   9GEC?5??6????            FROM ACCOUNT NUMBER PARAMATERS
   IAC   2   9LEC?8??9??10?           TO ACCOUNT NUMBER PARAMATERS
   FNC   2   9                        FULL ACCOUNT NUMBER
// END
*
// LOAD PROMP1                                   LOAD PRINT PROGRAM TO PRINT
// FILE NAME-APAPPRO,LABEL-AP.APPRO,DISP-SHR     SORTED VERSION OF FILE
// FILE NAME-APAPPROS
// RUN
```

Figure 20-5

Sample procedure PROMPT

```
*****************************************************************************
*  PROMPT.    THIS IS A SYSTEM/36 PROCEDURE THAT USES THE READ UNDER FORMAT   *
*             'RUF' TECHNIQUE FOR PASSING DATA FROM A PROMPT SCREEN TO A      *
*             WORKSTATION PROGRAM                                             *
*   1   SET DEFAULTS IN THE SCREEN HEADING BEFORE PROMPT SCREEN IS DISPLAYED  *
*   2   DISPLAY PROMPT SCREEN WITH DEFAULTS                                   *
*   3   LOAD WORKSTATION PROGRAM TO PROCESS PROMPT SCREEN                     *
*****************************************************************************
*
// EVALUATE P1-'1'                               SET SCREEN INPUT/OUTPUT ID
// IF ?TIME?>120000 EVALUATE P2-?TIME?PM         SET TIME TO AM OR PM
//   ELSE EVALUATE P2-?TIME?AM
// EVALUATE P3-'?USER?' P4-'?WS?'                SET HEADING DEFAULTS
// EVALUATE P5-01,2 P6-' 1' P7-'   1'            SET FROM ACCOUNT DEFAULTS
// EVALUATE P8-99 P9-99 P10-9999                 SET TO ACCOUNT DEFAULTS
*
// PROMPT MEMBER-PROMPTPM,FORMAT-SCREEN1,LENGTH-'1,8,,2,2,2,4,2,2,4,40',+
   PDATA-YES                                     DISPLAY PROMPT AND PASS DATA TO PROGRAM
*
// LOAD PROMPT                                   LOAD ON-LINE PROGRAM
// FILE NAME-APAPPRO,LABEL-AP.APPRO,DISP-SHR     PROCESS PROMPT SCREEN
// RUN
```

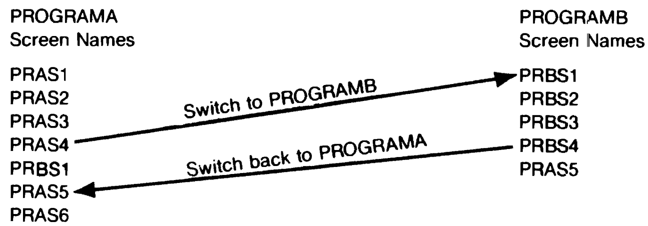
Figure 20-6

Sample procedure PROMRUF

```
*****************************************************************************
*  PROMRUF    THIS IS A SYSTEM/36 PROCEDURE THAT USES THE READ UNDER FORMAT   *
*             'RUF' TECHNIQUE FOR PASSING DATA FROM ONE ON-LINE INTERACTIVE   *
*             PROGRAM TO ANOTHER                                              *
*   1   SET OFF ALL SWITCHES                                                  *
*   2   CALL PROGRAMA                                                         *
*   3   IF SWITCH ONE IS ON, THEN CALL PROBRAMB                               *
*****************************************************************************
*
// TAG AGAIN                                 RESTART PROGRAMA
*
// SWITCH 00000000                           SET ALL SWITCHES OFF
*
// LOAD PROGRAMA                             LOAD AND RUN PROGRAM A
// FILE NAME-???????
// RUN
*
// IF SWITCH1-0 GOTO OUT                     IF SWITCH 1 IS OFF THEN EOJ
*
// LOAD PROGRAMB                             ELSE LOAD AND RUN PROGRAM B
// FILE NAME-?????
// RUN
*
// IF SWITCH2-1 GOTO AGAIN                   IF SWITCH 2 IS ON THEN RESTART PROGRAM A
*
// TAG OUT
```

Figure 20-7

Exit and entry screens for PROGRAMA and PROGRAMB



Creating Externally Described Workstation Files

by Gary Barrett

program by Rick Koenig

Code on diskette:
Procedure SFGRIO
RPG program SFGRIO
Screen format member SFGRIOFM
Message member SFGRIOM1

Many S/36 programmers use externally described disk files (i.e., disk file formats described outside the RPG program) to improve the consistency of field and file names, simplify documentation, and ease program maintenance. Externally described files are supported on the S/36 by the auto-report /COPY function, which inserts externally maintained S-specs into an RPG program before compilation.

But programmers find it difficult to use this capability for workstation files. A major deterrent is the lack of any simple method to create RPG I-specs and O-specs from the display format specifications compiled by $SFGR. The key benefits of external file definition depend on the use of a central "data dictionary" that provides a single source for format changes. But it is impossible to use just one source member for each screen format on the S/36 because of the way RPG screen I/O works with Workstation Data Management (WSDM).

WSDM uses the S- and D-specs compiled by $SFGR to create the physical layout of your displays and then merges that format with data from the workstation file buffer used by your RPG program under the control of your RPG I- and O-specs. Thus, at least *two* source members — the $SFGR screen format source member and the RPG source member — must be compiled for each workstation screen. IBM provides no easy way to link them together.

Granted, Screen Design Aid (SDA) provides an option to create a "skeleton" RPG program from screen format specifications. But the output from SDA is not in a format that can be used for externally described I/O without a considerable amount of work with SEU. And if the display changes, as frequently happens in interactive programs, the task of integrating the format changes into the RPG program usually is difficult and subject to error. The difficulty increases if input fields from displays have decimal values defined or if output data is edited to improve its appearance, such as with zero suppression.

The S/36 utility SFGRIO provides a more effective, easy-to-use alternative that fully supports externally described workstation screens. The utility processes your screen format S- and D-specs to generate the RPG I- and O-specs needed to manipulate that screen in your program. It creates separate input and output source members in the library you specify. These source members are accessible by using auto-report's /COPY function or by using the Include function of SEU.

Use of auto-report allows changes in the screen format source member to be reflected automatically in the RPG program simply by recompiling the program. In addition, the SFGRIO utility supports features SDA does not, including output field editing, the ability to combine array elements into a single array or multiple fields into a single field, and the ability to specify the number of decimal positions for a numeric input field.

SFGRIO uses the same S-specs as $SFGR, but accepts a modified D-spec format. The modified D-spec allows you to edit output fields in one of two ways: you can specify an edit code in either position 25 (normally the WSI edit field) or position 81 of the D-spec, or you can specify an edit work (following RPG conventions) by enclosing it within apostrophes beginning in position 81 of the D-spec. Note that the use of most edit codes or words on numeric fields that are both input and output may create programming problems, so the program will halt if an edit code other than Z is used for an I/O field. The operator then can choose to continue or cancel the programs. Also note that the length of the display format statements must be 96 or 120 to use column 81; this can be specified during SEU initiation.

The modified D-spec also supports combining array elements or multiple fields. You may frequently encounter several fields defined in the screen format member's D-specs that you want to use as elements of an array in an RPG program. In which case, you want to define one field name in the RPG program that encompasses all the fields in the D-specs that make up that array. The SFGRIO utility allows that to be done by coding each of those fields except the last one as @ followed by five blanks. These fields do not have I- or O-specs generated for them, but their lengths are accumulated into the length of the last field in the array, which has the same name as the array.

The utility also accepts a modification of the $SFGR D-spec that allows you to define, in column 24, the number of decimal positions for a numeric input field. The utility ignores column 24 if column 23 contains anything other than a Y. Further, the program assumes zero decimal places for defined numeric fields that have no entry in column 24. Note that if you use column 24 for decimal specification, you will be unable to condition numeric output on an indicator.

Using Utility SFGRIO

To execute utility SFGRIO, call procedure SFGRIO from a command screen. This procedure displays a prompt screen (Figure 20-8a) that lets you enter parameters that tell procedure SFGRIO where to get its input and what to do with its output. (The screen format member for the prompt screen is shown in Figure 20-8b)

Figure 20-8a

Procedure SFGRIO prompt screen

```
                              SFGRIO Procedure

Creates input/output source members from display format specifications.

INPUT

Display format source member name (-FM suffix)  .      .
Library name where display format exists .       .        .

OUTPUT

Output library name (where I/O modules will be placed)
Supply default field names if missing (Y,N)      ..
Halt before replacing duplicate source members? (Y,N)   .
Suppress Output End Positions? (Y,N)                     .




                    Command 7 - End Job, No Generation
```

Figure 20-9 shows modified S- and D-specs used as input to SFGRIO. For example, lines 19 and 29 contain edit words starting in positions 81; lines 5 and 17 contain an edit code in position 25. Lines 11 through 13 and 26 through 29 define two arrays.The first two parameters tell procedure SFGRIO (Figure 20-10) the screen format source member name and the name of the library in which it resides.

Parameter 3 specifies the library name where the generated RPG I- and O-specs are to be placed. You may want to set up a separate library to contain all your generated I- and O-specs so that there is never a conflict between the names procedure SFGRIO assigns to the RPG I- and O-spec source members and any production library members you have.

Parameter 4 allows you to specify that procedure SFGRIO should assign default field names if you have not included them in the D-specs. If you answer Y to this prompt, and procedure SFGRIO encounters a missing field name on the D-specs, the procedure assigns a field name of the form SF*xxxx*, where *xxxx* is a sequential number between 0001 and 9999. You can see that the default field names assigned to the generated RPG I- and O- specs are not very meaningful. Because the utility uses these field names in the input and output RPG statements it creates, it is best if you provide meaningful field names.

Parameter 5 allows you to request a halt before replacing a duplicate source member should a conflict arise between the source member name generated by SFGRIO and a source member already residing in the library you have specified for the output.

Parameter 6 allows you to suppress generation of the ending character position of the output fields by answering Y to this prompt. If you don't suppress this calculation, the program calculates the end position based on the field length specified in the D-spec.

For certain of these parameters, procedure SFGRIO sets defaults. Parameter 2, the source member library name, is initially set to the current library. Parameters 4, 5, and 6 are initially set to N. These defaults, which will appear on the prompt screen with the default values, can be overridden at the point of input.

After you have entered the necessary parameters, procedure SFGRIO issues an error message from message member SFGRIOM1 (Figure 20-11) if a parameter was entered incorrectly. This message member must be compiled as a level-one member (using the CREATE procedure) into the same library in which you have placed procedure SFGRIO and its associated program and screen member.

Upon successful validation of the input parameters, procedure SFGRIO uses the $MAINT utility to create a work file (?WS?.WORK) from the source member specified in parameter 1. Procedure SFGRIO then loads program SFGRIO (Figure 20-12), which reads the work file and generates two output files, IMEMBER and OMEMBER. These output files will contain the generated RPG input and output statements.

Program SFGRIO processes file ?WS?.WORK as an input sequential file. The program looks specifically for S- and D-specs. The program consists of two subroutines — SSPEC and DSPEC — that do all of the work. Subroutine SSPEC is called when an S-spec is read from file ?WS?.WORK and subroutine DSPEC is called when a D-spec is read from file ?WS?.WORK. All other records in the input file are ignored.

Subroutine SSPEC

Subroutine SSPEC (Figure 20-12, lines 75 through 100) generates the necessary information expected by $MAINT in output files IMEMBER and OMEMBER. Therefore, one of subroutine SSPEC's functions is to output the // COPY and // CEND records at the appropriate times in the output cycle. Subroutine SSPEC builds source member names for input and output by reading the format name in positions 7 through 14 of the S-spec record and then appending that name with an I for the input source member name and with a O for the output source member. The I and the O are left justified if the format name includes fewer than eight characters. If the format name is eight characters, the I and O are used in place of the eighth character. The name created in this manner is used as the object of the name parameter in the // COPY statement written to files IMEMBER and OMEMBER.

Subroutine SSPEC also writes a comment line as the first record after the // COPY record. This comment line provides valuable documentation information; it describes the format member from which the I- and O-specs are being created, the format member name, the library reference number for that source member, and the creation date and time.

Finally, subroutine SSPEC writes a record to file OMEMBER that contains the K8 keyword in positions 42 and 43 of the record, followed by the screen format name enclosed in apostrophes in positions 46 through 54 of the record. The format name is always left justified and padded with blank characters.

Subroutine DSPEC

Subroutine DSPEC (Figure 20-12, lines 107 through 222), which processes D-specs read from ?WS.WORK, is constructed in three sections. The first section (lines 107 through 123) handles screen constants within the D-specs that have no associated RPG I- or O-specs. The other two sections (lines 125 through 222) handle generation of field I- and O-specs to files IMEMBER and OMEMBER, respectively, for D-specs that specify input and/or output fields to be passed to or received from an associated RPG program.

Subroutine DSPEC first checks the D-spec for the presence of screen constants in positions 57 through 79. If a constant exists, the subroutine sets up a counter to provide for continuation of the constant onto the next D-spec. Constants are ignored in the I- and O-spec generation process. If no constants exist, the subroutine next checks to see whether the field is an input field, an output field, or both, and sets on indicators 20 and 21 for input and output fields, respectively.

Next, subroutine DSPEC checks the field name for @. If this is the field name, the field is defined to the program as a combined field. This means the field length is to be accumulated into a total field length and included in the input and/or output length calculation of the first valid field that follows. If the field name is not @, it is treated as a standard input/output field.

For input fields, subroutine DSPEC computes the starting and ending position of the field in the input buffer and, if it is a numeric field, determines the number of decimal positions. Subroutine DSPEC compensates for the extra digit defined on the D-spec for signed numeric fields. Subroutine DSPEC also supplies the field name if that is missing. Finally, subroutine DSPEC writes a record to file IMEMBER describing the input field in RPG I-spec input format, using EXCPT name output.

For an output field, subroutine DSPEC first determines whether it is a message type output field and, if so, whether the message is sent from the program or from a message member or is a constant in the D-spec. If the output is a constant or from a message member (identified by a MIC number), no further processing is necessary. If the output message is generated by the associated RPG program, its output length will be set automatically to six characters, regardless of the field length specified in the D-spec.

When program SGFRIO finishes processing all S- and D-specs in file ?WS?.WORK, it returns control to procedure SFGRIO. (At this point, each source member in files IMEMBER and OMEMBER is bracketed by a // COPY statement and a // CEND statement so that files IMEMBER and OMEMBER look as expected by the $MAINT utility.) Procedure SFGRIO then uses the $MAINT utility a second time to create source members in the library specified for each I/O member created in files IMEMBER and OMEMBER, respectively. The source members created in this manner are now available for inclusion in the appropriate RPG programs.

The auto-report copy function provides an excellent way to include source members in your RPG programs. For example, Figure 20-13 is a sample RPG program that uses the /COPY auto-report statement (lines 11 and 26) to include the sample source members shown in Figures 20-14a and 20-14b. Notice in Figures 20-14a and 20-14b that file and record identification entries do not exist for either input or output source members. You need to supply these in your RPG program, as you normally would.

As you implement the SFGRIO utility, be aware that the use of some of the techniques described will result in warning errors when the format member is compiled. But these warning errors do not affect the usability of the load member produced by the compilation.

Figure 20-8b

Screen format member SFGRIOFM

```
S*..1... :. ..2.. ... .3..  ....4 ..: ..5.. ..  6      7   :.  8
SSFGRIO              Y                                  G
D          16 132Y                          Y       CSFGRIO Procedure
D          71 3 4Y                                  CCreates input/output soX
Durce members from display format specifications.
D           5 5 4Y                          Y       CINPUT
D          60 7 4Y                                  CDisplay format source mX
Dember name (-FM suffix).........      .
DPARAM1     8 76501 YB   91     Y     91  Y
D          60 8 4Y                                  CLibrary name where dispX
Dlay format exists . ... ..   . . .. .
DPARAM2     8 86502 Y    92     Y     92  Y
D           610 4Y                          Y       COUTPUT
D          6012 4Y                                  COutput library name (whX
Dere I/O modules will be placed) . .
DPARAM3     8126503 Y    93     Y     93  Y
D          6013 4Y                                  CSupply default field naX
Dmes if missing (Y,N) .. .....   ..  .
DPARAM4     1136504 Y    94     Y     94  Y          0
DPAD04      71367Y  Y          Y    Y
D          6014 4Y                                  CHalt before replacing dX
Duplicate source members? (Y,N)    ...
DPARAM5     1146505 Y    95     Y     95  Y          0
DPAD05      71467Y  Y          Y    Y
D          6015 4Y                                  CSupress Output End PosiX
Dtions? (Y,N) .... ... ..   ..   . .
DPARAM6     1156506 Y    96     Y     96  Y          0
DPAD06      71567Y  Y          Y    Y
DMSGMIC    7522 407                   97            M
D          342423Y                                  CCommand 7 - End Job, NoX
D Generation
```

Figure 20-9

Modified S- and D-specs for the SFGRIO utility

```
S*..1         2         3         4         5         6         7          8
SSFTESTA              Y                                   G
DSCRTYP     1 1 4Y  Y          Y     Y            CA
D          15 126Y                                CAccount Inquiry
DUDATE      8 159Y Y
D          66 3 4Y                                CThis allows inquiry intX
Do any of the active accounts on the system
D           8 5 7Y                                CAccount#
DACCTNO     6 516Y  Y            Y         Y
D           7 524Y                                CThru Dt
D@          2 532Y  YN
D@          2 535Y  YN
DTHRUDT     2 538Y  YN
D           4 542Y                                CName
DACTNAM    25 547Y
D          15 7 7Y                                CCurrent Balance
DACTBAL    13 723Y
D          18 740Y                                CLast Activity Date
DLACTDT     8 759Y                                                         '  /  / 0'
D          27 9 7Y                                CPrevious Quarterly BalaX
Dnces
D          1110 7Y                                C1st Quarter
D          111022Y                                C2nd Quarter
D          111037Y                                C3rd Quarter
D          111052Y                                C4th Quarter
D@         1311 7Y
D@         131122Y
D@         131137Y
DQTR       131152Y                                                         ' ,   ,  0   -'
D          152326Y                                CCmd 7 - End Job
```

Figure 20-10

Procedure SFGRIO

```
* PROCEDURE - SFGRIO
* FUNCTION - CREATE DISPLAY FORMAT I/O SOURCE MEMBERS
*
* PARAMETER SUMMARY
* PARAM 1 - SCREEN FORMAT MEMBER NAME
* PARAM 2 - SCREEN FORMAT LIBRARY NAME
* PARAM 3 - LIBRARY NAME FOR GENERATED I/O MODULES
* PARAM 4 - SUPPLY DEFAULT FIELD NAMES- Y,N
* PARAM 5 - HALT ON REPLACE OPTION- Y,N
* PARAM 6 - SUPRESS OUTPUT END LOCATIONS - Y,N
* PARAM 7 - MESSAGE MIC FOR ERRORS
* PARAM 8 - DUMMY RESPONSE PARAM
*
* LDA USAGE
*   1-6  DEFAULT FIELD NAME (SF####)
*   5-6  NUMBER OF DEFAULT FIELD NAMES GENERATED (AT EOJ)
*   7-7  INPUT MEMBER NAME SUFFIX (DEFAULT IS 'I')
*   8-8  OUTPUT MEMBER NAME SUFFIX (DEFAULT IS 'O')
*
* EXTERNAL SWITCH USAGE
*  U1 - SUPRESS INPUT FIELD LOCATIONS IN PROGRAM
*  U8 - AT LEAST ONE I/O EDIT OTHER THAN 'Z' WAS FOUND
* NOTE - SWITCHES U1-U8 ARE MAPPED TO 91-98 INITIALLY FOR VALIDATION PURPOSES
*
* SET DEFAULT PARAMETER VALUES
// SWITCH 00000000   ?2'?CLIB?'? ?4'N'? ?5'N'? ?6'N'? ?7'9999U1'?
// MEMBER USER1-SFGRIOM1
// TAG START
// PROMPT MEMBER-SFGRIOFM,FORMAT-SFGRIO,UPSI-YES
// SWITCH 00000000  ?7F''?
// IF ?CD?/2007   RETURN
* PERFORM PARAMETER VALIDATION
// IF  ?1?/                        SWITCH 1XXXXX1X ?7'0100U1'? . SCR NAME MISSING
```

```
// IF  ?2?/                          SWITCH X1XXXX1X ?7'0200U1'? , LIBR NAME MISSING
// IFF ?2?/  IFF DATAF1-?2?          'SWITCH X1XXXX1X ?7'0201U1'? , LIBR NAME NOT ON DISK
// IFF ?2?/  IFF LOAD-'#PTFLOG,?2?'  SWITCH X1XXXX1X ?7'0202U1'? , P2 NOT A LIBR
// IF  ?7?/  IFF SOURCE-'?1?,?2?'    SWITCH 1XXXXX1X ?7'0101U1'? , P1/SCR NAME NOT IN LIBR
// IF  ?3?/                          SWITCH XX1XXX1X ?7'0300U1'? , P3 MISSING
// IFF ?3?/  IFF DATAF1-?3?          SWITCH XX1XXX1X ?7'0301U1'? , P3 NOT ON DISK
// IFF ?3?/  IFF LOAD-'#PTFLOG,?3?'  SWITCH XX1XXX1X ?7'0302U1'? , P3 NOT A LIBR
// IFF ?4?/Y IFF ?4?/N               SWITCH XXX1XX1X ?7'0400U1'? , P4 NOT Y/N
// IFF ?5?/Y IFF ?5?/N               SWITCH XXXX1X1X ?7'0500U1'? , P5 NOT Y/N
// IFF ?6?/Y IFF ?6?/N               SWITCH XXXXX11X ?7'0600U1'? , P6 NOT Y/N
* IF PARAM 7 IS NOT NULL (OR SWITCH 7 ON) REPROMPT WITH ERRORS
// IFF ?7?/   GOTO START
*----- EXECUTE -----
// * 'SFGRIO PROCEDURE EXECUTING'
// * 'CREATING ?1? I/O MEMBERS FROM ?2?, OUTPUT TO ?3?...'
// * '    DEFAULT NAMES-?4?  HALT BEFORE REPLACE-?5?  SUPRESS END POS-?6?'
// LOCAL OFFSET-1,DATA-'SF0000IO '
// SWITCH 00000000          , ?8F''? MAKE SURE PARAM8 IS NULL
*
* CONVERT PARAM 5 FROM Y/N TO RETENTION CODE P/R
// IF ?5?/Y SWITCH XXXXXXXX  ?5F'P'?
// IF ?5?/N SWITCH XXXXXXXX  ?5F'R'?
*
* IF PARAM 6 IS Y, SET SWITCH 1 ON TO SUPRESS OUTPUT POSITIONS
// IF ?6?/Y  SWITCH 1XXXXXXX
*
*  EXECUTE $MAINT COPY
// LOAD $MAINT
// FILE NAME-LIBRFILE,LABEL-?WS?.WORK,RECORDS-1000,RETAIN-J,EXTEND-500
// RUN
// COPY FROM-?2?,TO-DISK,FILE-LIBRFILE,RECL-120,NAME-?1?,LIBRARY-S,BASIC-YES,SVATTR-YES
// END
*
// LOAD SFGRIO
// FILE NAME-LIBRFILE,LABEL-?WS?.WORK
// FILE NAME-IMEMBER,RECORDS-1000,RETAIN-J,EXTEND-500
// FILE NAME-OMEMBER,RECORDS-1000,RETAIN-J,EXTEND-500
// RUN
*
* IF DEFAULT FIELD NAMES SUPPLIED AND PARAM 4 WAS 'N' - ISSUE MSG & RETURN
// IFF ?L'3,4'?/0000 IF ?4?/N IF ?8R'9000'?/?8?    RETURN
* IF DEFAULT FIELD NAMES SUPPLIED AND PARAM 4 WAS 'Y' - DISPLAY # & CONTINUE
// IFF ?L'3,4'?/0000 IF ?4?/Y * '?L'3,4'? DEFAULT FIELD NAMES WERE SUPPLIED BY SFGRIO'
*
* PROMPT FOR CONTINUE OPTION IF SWITCH 8 IS ON. SEE MESSAGE MIC 9001.
// IF SWITCH8-1  IFF ?R'9001'?/Y   RETURN
*
// * 'I/O MEMBERS BEING COPIED TO LIBRARY ?3?'
// LOAD $MAINT
// FILE NAME-IMEMBER,UNIT-F1
// FILE NAME-OMEMBER,UNIT-F1
// RUN
// COPY TO-?3?,FROM-DISK,FILE-IMEMBER,RETAIN-?5?
// COPY TO-?3?,FROM-DISK,FILE-OMEMBER,RETAIN-?5?
// END
```

Figure 20-11

Parameter validation messages from message member SFGRIOM1

```
SFGRIOM1,1
* PARAMETER VALIDATION MESSAGES
0100      DISPLAY FORMAT SOURCE MEMBER NAME MISSING; MUST BE SPECIFIED
0101      DISPLAY FORMAT SOURCE MEMBER NAME IS NOT IN THE INPUT LIBRARY
0200             INPUT LIBRARY NAME MISSING, THIS IS REQUIRED
0201          INPUT LIBRARY NAME SPECIFIED DOES NOT EXIST ON THE DISK
0202             INPUT LIBRARY NAME SPECIFIED IS NOT A LIBRARY
0300           OUTPUT LIBRARY NAME IS MISSING; THIS IS REQUIRED
0301         OUTPUT LIBRARY NAME SPECIFIED DOES NOT EXIST ON THE DISK
0302           OUTPUT LIBRARY NAME SPECIFIED IS NOT A LIBRARY
0400     DEFAULT-FIELD-NAMES OPTION IS BLANK OR INVALID (MUST BE Y OR N)
0500     HALT-BEFORE-REPLACE OPTION IS BLANK OR INVALID (MUST BE Y OR N)
0600     OUTPUT-END-POSITION OPTION IS BLANK OR INVALID (MUST BE Y OR N)
* MESSAGES FOR THE PROCEDURE
9000 MISSING FIELD NAMES FOUND, DEFAULTS NOT SELECTED; PRESS ENTER TO CANCEL.
9001 AN I/O FIELD FOUND WITH EDIT CODE OTHER THAN "Z"; CONTINUE? (Y,N)
9999
```

Figure 20-12

Program SFGRIO

```
*        1         2         3         4         5         6         7         8
0001 H     064                      B       1                             SFGRIO
0002 H*  THIS PROGRAM CREATES RPG 'I' AND 'O' SPECIFICATIONS FROM
0003 H*  DISPLAY FORMAT 'S' AND 'D' SPECIFICATIONS
0004 H*
0005 H*  INDICATOR USAGE SUMMARY
0006 H*  01 - S SPECIFICATION INPUT
0007 H*  02 - D SPECIFICATION INPUT
0008 H*  03 - // COPY STATEMENT FROM $MAINT
0009 H*  04 - INPUT CATCH-ALL
0010 H*  20 - D-FIELD IS INPUT TYPE
0011 H*  21 - D-FIELD IS OUTPUT TYPE
0012 H*  40 - GENERAL USE TO CONTROL LOGIC
0013 H*  90 - SUPRESS '// CEND' ON FIRST S-SPEC
0014 H*  U1 - SUPRESS OUTPUT END POSITION IF ON
0015 H*  U8 - I/O EDIT OTHER THAN 'Z'; ISSUE WARNING
0016 H*
0017 FLIBRFILEIPE F1200 120    2       DISK
0018 FIMEMBER O   F 960  96    2       DISK
0019 FOMEMBER O   F 960  96    2       DISK
0020 E                    SF          8  1               —SCR FMT NAME
0021 ILIBRFILENS  01   6 CS   7NC*
0022 I* S-SPEC
0023 I                                        7  14 SFMT
0024 I                                        7  14 SF
0025 I        NS  02   6 CD   7NC*
0026 I* D-SPEC
0027 I                                        7  12 DNAM                        NAME
0028 I                                       15  180DLEN                        LENGTH
0029 I                                       23  23 DOUT                        OUTPUT
0030 I                                       24  24 DDEC                        #DECIMALS
0031 I                                       25  25 DEDC                        EDIT CODE
0032 I                                       26  26 DINP                        INPUT TYPE
0033 I                                       27  27 DTYP                        DATA TYPE
0034 I                                       56  56 DCON                        CONST/MSG FLAG
0035 I                                       57  79 DATA                        CONSTANT/MIC
0036 I                                       81  81 EDIT                        ALT EDIT CODE
0037 I                                       81 106 WORD                        EDIT WORD
0038 I        NS  03   1 C/   2 C/   4 CC
0039 I                                       24  66 MEMB
0040 I* CATCH-ALL FOR OTHER RECORD TYPES
0041 I        NS  04
0042 I/SPACE
0043 I            DS
0044 I* THIS DATA STRUCTURE CONTAINS MISCELLANEOUS VARIABLES
0045 I                                        1  48 LOC
0046 I                                        1   40ISTART
0047 I                                        5   80IEND
0048 I                                        9  1200END
0049 I                                       13  160INUM
0050 I                                       17  2000NUM
0051 I                                       21  240FSIZE
0052 I                                       25  280IADD
0053 I                                       29  3200ADD
0054 I                                       33  40 SFMTI
0055 I                                       41  48 SFMTO
0056 I            DS
0057 I                                        1   8 SF
0058 I                                        1   8 SFDS
0059 I/SPACE
0060 I           UDS
0061 I                                        1   6 SFXXXX                      DEFAULT FLD NAME
0062 I                                        5   60XXXX                        NAME COUNTER
0063 I                                        7   7 ISUFFX                      INPUT SUFFIX
0064 I                                        8   8 OSUFFX                      OUTPUT SUFFIX
0065 C* CALCULATION MAINLINE
0066 C*
0067 C    01                EXSR SSPEC
0068 C    02                EXSR DSPEC
0069 C*
0070 CLR                    EXCPTCEND                       —ADD // CEND
```

```
C*
C* ************************************************************
C           SSPEC     BEGSR                                * SSPEC *
C* ************************************************************
C      90             EXCPTCEND                        —ADD // CEND
C     N90             SETON                      90    —ONE TIME SWITCH
C*
C                     MOVE *BLANK    LOC               —CLEAR VARIABLES
C* CREATE I/O MEMBER NAMES FROM FORMAT NAME AND SUFFIXES
C                     Z-ADD1         X       10
C*                                               >><<--
C          *BLANK     LOKUPSF,X                      40—FIRST BLANK
C     N40             Z-ADD8         X                 —OVERRIDE LAST BYTE
C                     MOVE ISUFFX    SF,X              —INPUT SUFFIX
C                     MOVE SFDS      SFMTI             —MAKE INPUT NAME
C                     MOVE OSUFFX    SF,X              —OUTPUT SUFFIX
C                     MOVE SFDS      SFMTO             —MAKE OUTPUT NAME
C*
C* OUTPUT IMEMBER HEADER RECORDS
C                     EXCPTICOPY                       —// COPY (I)
C           INUM      ADD  1         INUM              —INPUT LINE#
C                     EXCPTICOMM                       —I* COMMENT
C* OUTPUT OMEMBER HEADER RECORDS
C                     EXCPTOCOPY                       —// COPY (O)
C           ONUM      ADD  1         ONUM              —OUTPUT LINE#
C                     EXCPTOCOMM                       —O* COMMENT
C           ONUM      ADD  1         ONUM              —OUTPUT LINE#
C                     EXCPTOFORM                       —K8 'FORMATNM'
C*
C                     ENDSR
C*
C* ************************************************************
C           DSPEC     BEGSR
C* ************************************************************
C*
C* CHECK FOR PENDING CONSTANT                    >><<--
C           FSIZE     COMP *ZERO                 40    —PENDING LARGE CONSTANT
C      40   FSIZE     SUB  73        FSIZE             —SUBTRACT ANOTHER 73
C      40             GOTO DSPEC9                      —EXIT
C*                         ---
C           FSIZE     SUB  FSIZE     FSIZE             —CLEAR IT
C*
C* CHECK FOR MISSING CONSTANT CODE               >><<--
C           DATA      COMP *BLANK                40    —IF CONSTANT DATA
C      40   DCON      COMP *BLANK                    40—AND 'C' IS MISSING
C      40             MOVE 'C'       DCON              —THEN ADD IT
C*
C* IF CONSTANT AND SIZE IS OVER 23, SET UP
C* THE FSIZE FIELD TO HANDLE IT                  >><<--
C           DCON      COMP 'C'                       40—CONSTANT
C      40   DLEN      COMP 23                    40    —OVER ONE LINE
C      40   DLEN      SUB  23        FSIZE             —AMT LEFT
C      40             GOTO DSPEC9
C*                         ---
C* SET I/O INDICATORS  20-INPUT  21-OUTPUT       >><<--
C           DINP      COMP 'Y'                       20—INPUT IF EQ
C           DOUT      COMP 'Y'                   21  21—OUTPUT IF Y OR GREATER
C*  20 AND 21 ARE THE ONLY IND  USED WITHOUT RESETTING PRIOR TO USE
C*
C* IF NAME IS '@', ROLL IN ADDERS AND EXIT       >><<--
C           DNAM      COMP '@      '                 40—ROLL UP FLD NAME
C    20 40  IADD      ADD  DLEN      IADD              —INCR INPUT LENGTH
C    21 40  OADD      ADD  DLEN      OADD              —INCR OUTPUT LENGTH
C      40             GOTO DSPEC9                      —EXIT
C*                         ---
C* PROCESS INPUT DATA, IF NO INPUT, SKIP TO OUTPUT
C  N20                GOTO DSPEC5
C*                         ---
C* SET DECIMAL FIELD                             >><<--
C           DOUT      COMP 'Y'                   4040  —IF NOT OUTPUT
C      40             MOVE *BLANK    DDEC              —FORCE BLANK
C* ADJUST FIELD LENGTH FOR SIGNED NUMERICS       >><<--
C           DTYP      COMP 'S'                       40—IF SIGNED NUM
C      40   DLEN      SUB  1         DLEN              —LENGTH-1
C* IF NUMERIC AND NO #DECIMALS, SET DEFAULT      >><<--
```

```
C           DTYP      COMP 'S'                      40-IF SIGNED NUM
C      N40  DTYP      COMP 'D'                      40-OR DECIMAL
C      N40  DTYP      COMP 'N'                      40-OR NUMERIC
C       40  DDEC      COMP *BLANK                   40-AND NO DEC. SIZE
C       40            MOVE 'O'       DDEC             -ASSUME ZERO
C*
C* SET INPUT START/END POSITIONS
C           IEND      ADD  1         ISTART
C           IEND      ADD  DLEN      IEND
C           IEND      ADD  IADD      IEND
C                     Z-ADD*ZERO     IADD
C           INUM      ADD  1         INUM
C*
C* SUPPLY FIELD NAME IF MISSING                   >><<--
C           DNAM      COMP *BLANK                   40-NO NAME
C       40  XXXX      ADD  1         XXXX             -INCR COUNTER
C       40            MOVE SFXXXX    DNAM             -SUPPLY DEFAULT
C*  WRITE RECORD TO FILE 'IMEMBER'
C                     EXCPTIDATA                      -WRITE INPUT DATA
C*          ---
C           DSPEC5    TAG
C*          ---
C* IF NO OUTPUT, EXIT   (IND  21 - OUTPUT, SET PREVIOUSLY)
C  N21                GOTO DSPEC9                     -EXIT
C*                    ---
C* IF OUTPUT CONSTANT, EXIT                       >><<--
C           DCON      COMP 'C'                      40-CONSTANT
C       40            GOTO DSPEC9                     -EXIT
C*                    ---
C* IF NOT A MSG FIELD, SKIP MIC PROCESSING        >><<--
C           DCON      COMP 'M'                      40-MSG/MIC FLAG
C      N40            GOTO DSPEC6                     -NO, SKIP
C*                    ---
C* IF MIC IS GIVEN, NO OUTPUT NEEDED; EXIT         >><<--
C           DATA      COMP *BLANK                 40    -MSG MIC PRESENT,
C       40            GOTO DSPEC9                     -SO EXIT
C*                    ---
C* IF MSG-TYPE AND NO MIC, IT IS PROGRAM-SUPPLIED.
C* REGARDLESS OF D-SPEC LENGTH, OUTPUT BUFFER SIZE IS 6
C                     Z-ADD6         DLEN             -SET LENGTH TO 6
C*          ---
C           DSPEC6    TAG
C*          ---
C* IF EDIT CODE IS BLANK, SKIP TO OUTPUT          >><<--
C           DEDC      COMP *BLANK                 40    -EDIT CODE PRESENT
C       40  EDIT      COMP *BLANK                   40-& COL.81 BLANK
C       40            MOVE *BLANK    WORD             -CLEAR EDIT WORD
C       40            GOTO DSPEC8                     -NO EDIT CD/WORD
C*                    ---
C* IF EDIT IS NOT QUOTE, ASSUME EDIT CODE         >><<--
C           EDIT      COMP ''''                     40-APOSTROPHE
C      N40            MOVE EDIT      DEDC             -MAKE IT AN EDIT CODE
C      N40            MOVE *BLANK    WORD             -CLEAR 'WORD'
C*
C* IF 20 (INPUT) AND EDIT CODE SPECIFIED, OTHER THAN Z,
C* PLACE A FLAG IN THE LDA TO WARN OPERATOR.      >><<--
C   20      EDIT      COMP *BLANK                 40    -NON-BLANK
C   20 40   EDIT      COMP 'Z'                    4040  -AND NOT 'Z'
C   20 40             SETON                       U8    -INPUT/NOT Z-ISSUE WARNING
C*
C*          ---
C           DSPEC8    TAG
C*          ---
C* SET OUTPUT END POSITION
C           OEND      ADD  DLEN      OEND
C           OEND      ADD  OADD      OEND
C                     Z-ADD*ZERO     OADD
C           ONUM      ADD  1         ONUM
C*
C* SUPPLY FIELD NAME IF MISSING                   >><<--
C           DNAM      COMP *BLANK                   40-NO NAME
C       40  XXXX      ADD  1         XXXX             -INCR COUNTER
C       40            MOVE SFXXXX    DNAM             -SUPPLY DEFAULT
C*  WRITE RECORD TO FILE 'OMEMBER'
C                     EXCPTODATA                      -WRITE OUTPUT DATA
```

```
C*           ---
C            DSPEC9    ENDSR
C*           ---
O* --------------------------
OIMEMBER E                 ICOPY
O                                   23 '// COPY LIBRARY-S,NAME-'
O                          SFMTI    31
O* --------------------------
OIMEMBER E                 ICOMM
O                          INUM      4
O                                   26 'I*  INPUT FOR FORMAT'
O                          SFMT     34
O                                   47 'FROM MEMBER '
O                          MEMB     91
O* --------------------------
OIMEMBER E                 IDATA
O                          INUM      4
O                                    6 'I'
O                          ISTARTZ  47
O                          IEND  Z  51
O                          DDEC     52
O                          DNAM     58
O* --------------------------
OIMEMBER E                 CEND
O                                    7 '// CEND'
O* --------------------------
OOMEMBER E                 OCOPY
O                                   23 '// COPY LIBRARY-S,NAME-'
O                          SFMTO    31
O* --------------------------
OOMEMBER E                 OCOMM
O                          ONUM      4
O                                   26 'O* OUTPUT FOR FORMAT '
O                          SFMT     34
O                                   47 'FROM MEMBER '
O                          MEMB     91
O* --------------------------
OOMEMBER E                 OFORM
O                          ONUM      4
O                                    6 'O'
O                                   54 'K8 ''          '''
O                          SFMT     53
O* --------------------------
OOMEMBER E                 ODATA
O                          ONUM      4
O                                    6 'O'
O                          DNAM     37
O                          DEDC     38
O                    NU1   OEND  Z  43
O                          WORD     70
O* --------------------------
OOMEMBER E                 CEND
O                                    7 '// CEND'
```

Figure 20-13

Sample program using auto-report

```
*         1         2         3    . 4    ... 5 .       6         7         8
U
H     064                             B         1      01                       SFTEST
H*  SFTEST IS A SIMPLE WORKSTATION PROGRAM WHICH DEMONSTRATES
H*  THE USE OF THE SFGRIO PROGRAM & PROCEDURE
FWORKSTN CP  F    1920               WORKSTN
F                                                 KID    WS
F                                                 KFMTS  SFTESTFM
FACCTMASTIC  F 256 256R06AI      1 DISK
E                     QTR          4  9 2                QUARTERLY BALANCES
IWORKSTN NS  01   1 CA
I/COPY DEMOLIBR,SFTESTAI
IWORKSTN NS  02
I* CATCH-ALL
IACCTMASTNS
I                                         2   7 ACCTNO
I                                         8  130THRUDT
I                                        14  38 ACTNAM
```

```
I                                         39  472ACTBAL
I                                         48  530LACTDT
I                                         54  89 QTR
C    02                  GOTO ENDDET
C    01NKG    ACCTNO     CHAINACCTMAST
C    01 KG               SETON                      LR
C             ENDDET     TAG
OWORKSTN D        NLR
O/COPY DEMOLIBR.SFTESTAO
```

Figure 20-14a

I-specs generated by SFGRIO

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
I*  INPUT FOR FORMAT SFTESTA  FROM MEMBER  SFTESTFM,REF-000011,DATE-84/09/30,TIME-1046
I                                        1    1 SCRTYP
I                                        2    7 ACCTNO
I                                        8   130THRUDT
```

Figure 20-14b

O-specs generated by SFGRIO

```
*... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
O* OUTPUT FOR FORMAT SFTESTA  FROM MEMBER  SFTESTFM,REF-000011,DATE-84/09/30,TIME-1046
O                                      K8 'SFTESTA '
O                          UDATE Y     8
O                          ACCTNO     14
O                          THRUDT     20
O                          ACTNAM     45
O                          ACTBALJ    58
O                          LACTDT     66 '  /  / 0'
O                          QTR       118 ' .   . 0.  -'
```

Creating S/36 Help Screens on a PC

by John W. Warns

Code on diskette:
Procedure HPMAKE
RPG program HPMAKE

Traditionally, the production of help screens has been the responsibility of the programmer, but letting programmers control this user service has not always met with sterling success. Programmers sometimes produce help screens that are too technical in nature or that fail to address all user needs. Frequently, the best way to compensate for these shortcomings is to let users enhance existing on-line help text themselves.

Until now, this has meant training users in such utilities as SDA or devoting valuable programming time to the translation of user definitions into S- and D-specs (either manually or through SDA). But there is another way to accomplish this task. Why not let users create and maintain help screens on their PCs within the confines of their favorite word processing program? You can then merge the user's help information with the current help database.

The ability to set up help text this way has been available since we have had the ability to connect the S/36 and the PC through emulation. The process is straightforward:

1. Create a predefined shell document for the users' use with their word processing package.

2. Convert the shell document (after the user has entered help text) to a file portable to the S/36 (such as a standard ASCII text file).

3. Transfer this file to a S/36 virtual disk.

4. Convert the virtual disk file to a S/36 file via PC Support/36.

5. Run program HPMAKE, which creates a file of S-and D-specs.

6. Copy this file to a source member.

7. Compile the screens using the S/36 FORMAT command.

The Shell Document

The successful implementation of this procedure depends on the creation of a stable, predictable user-input shell document. The help screen in Figure 20-15 (constructed in WordPerfect) illustrates points you should consider when creating your shell document:

1. The shell document begins with the word Screen in positions 4 through 9 of the document. When program HPMAKE finds this word in these positions, it attempts to process subsequent lines of help text.

2. The eight-character help screen name appears in columns 18 through 25. The first six characters of the name (the first time it is encountered) define the source member name. You can instruct program HPMAKE to append a suffix to the help screen load member's name. Remember that the eight-character help screen title eventually becomes the help format name (i.e., S-spec name); therefore, it must adhere to $SFGR help-format naming conventions (i.e., the name must be eight characters long, begin with an alphabetic character, and end in two numeric digits between 00 and 99). Remember, too, that you must reference these help screen format names in your applications program's screen format member H-specs.

3. Each line of help text begins with a two-digit line number in positions 1 and 2. The help text itself must be in positions 5 through 82. Lines without a two-digit line number are ignored.

You can use these ignored lines as design guides for the user in preparing the help screens. For example, borders on the top, bottom, and sides lend spatial guidance to the user when designing the screen. Note that I've defined only 23 lines and 78 characters on the sample shell document. You may design your form with 24 lines and 80 characters, but if you do so, don't forget that position 1 of line 1 must remain blank (this position is reserved by the SSP).

You may string together any number of copies of this shell document (each with a different name suffix) to create the variety of screens usually needed to provide documentation for any single program.

The Export

After the user composes the help text in the shell document, you are ready to transfer the document to the S/36 and convert it; for your purposes, you need to create a standard ASCII text file. Almost all word processing applications provide the capability to export material in this way. In WordPerfect, you run the convert utility, which prompts you for the name of the file to be converted and the new name of the converted file. Next, you select option 1, WordPerfect to another format, and then you select option 7, ASCII text file.

The Transfer

The device that logically connects your PC and the S/36 is the virtual disk, which is actually a direct file located on the S/36 hard drive. Almost all existing DOS commands may be used with a virtual disk. You can copy your ASCII help text file from the PC to the virtual disk using the DOS copy command.

To copy the virtual disk file to a S/36 file, use PC Support/36's PCU (Personal Computer Utility) procedure, which lets you perform a number of related tasks, including creating a virtual disk, copying S/36 source code and/or procedures, and copying data files to and from a virtual disk.

You must provide a number of parameters to copy a virtual disk file on the S/36; of these, the one that is not obvious is the record length for the new file. This variable depends on how you created the user's shell document. The example provided (Figure 20-15) has a record length of 84 characters, so you enter 84 into this parameter. After completing this task, you will have a copy of the PC ASCII file on your S/36 as a standard EBCDIC sequential file.

The Conversion

Program HPMAKE (Figure 20-16) converts the file you transferred from the PC into a $MAINT file containing S- and D-specs (Figure 20-17). You can use the TOLIBR procedure to copy this file into a library source member.

Program HPMAKE generates three basic types of screen specifications from the input text file: S-specs, D-specs, and D-spec continuation lines. The array SPEC contains the "prototypes" of these statements, as well as the // COPY and // CEND statements required in a $MAINT file.

Every time program HPMAKE encounters a screen header line, it outputs an S-spec. Subsequent lines having a line number in the range 01 to 24 get output as D-specs and D-spec continuation lines.

You can automate the entire process on the S/36 side with the procedure shown in Figure 20-18. This procedure first clears the two work files

HELPS and SDSPECS from the disk. Then the file HELPS.SDF (a generic file I pass all help screens to) is accessed from the virtual disk (named PCV) and copied to the S/36 as a file called HELPS. Next, program HPMAKE converts the shell document to a file called SDSPECS. The last statement copies the file SDSPECS to the default library.

The PC side may also be automated with the .BAT file shown in Figure 20-19, which assigns the virtual disk and copies the extracted file (using a generic name that matches on both sides).

Letting users enhance their on-line help facilities reduces your application maintenance effort and can help draw users into a deeper understanding of their applications, all of which puts users closer to controlling their data processing destiny.

Figure 20-15
HLPTXT01 screen

```
*       1         2         3    .  4         5         6         7         8

  Screen Name - HLPTXT01

           1         2         3         4         5         6         7
  12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
  -----------------------------------------------------------------------------
01 |                        HELP SCREEN CREATION DEMO SCREEN                        |
02 |                                                                                |
03 |                                                                                |
04 |                                                                                |
05 |                                                                                |
06 |                   To be, or not to be: that is the question.                   |
07 |                   Whether 'tis nobler in the mind to suffer                    |
08 |                   The slings and arrows of outrageous fortune,                 |
09 |                   Or to take arms against a sea of troubles,                   |
10 |                   And by opposing end them?  To die. to sleep,                 |
11 |                   No more; and, by a sleep to say we end                       |
12 |                   The heart-ache and the thousand natural shocks               |
13 |                   That flesh is heir to, 'tis a consummation                   |
14 |                   Devoutly to be wish'd.  To die, to sleep;                    |
15 |                   To sleep· perchance to dream: ay, there's the rub;           |
16 |                                                                                |
17 |                                                                                |
18 |                                                HAMLET Act 3 Scene 1            |
19 |                                                William Shakespeare             |
20 |                                                                                |
21 |                                                                                |
22 |                                                                                |
23 |             PLEASE PRESS THE ENTER KEY TO RETURN TO PROCESSING                 |
  -----------------------------------------------------------------------------
  12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
           1         2         3         4         5         6         7
```

Figure 20-16
Program HPMAKE

```
*        1 .       2 .       3         4 .       5         6         7 . . .   8
0001 H       64                      B       1                            HPMAKE
0002 F**************************************************************
0003 F*              WRITTEN BY.         J  W  WARNS                 *
0004 F**************************************************************
0005 F**************************************************************
0006 F*  P R O G R A M    D E S C R I P T I O N                      *
0007 F**************************************************************
```

```
F*
F*  CONVERT TEXT FOR HELP SCREENS INTO S & D SPECIFICATIONS
F*
F*********************************************************************
F*   I N D I C A T O R   S U M M A R Y                               *
F*-------------------------------------------------------------------*
F*                                                                   *
F*********************************************************************
F/SPACE
FHELPS   IPE F  84  84            DISK
FSDSPECS O   F  80  80            DISK                      A
E/SPACE 2
E***********************************************************
E* E X C E P T I O N  S P E C I F I C A T I O N S          *
E***********************************************************
E                    SPEC    1   5 80
I/SPACE 2
I***********************************************************
I* I N P U T  S P E C I F I C A T I O N S                  *
I***********************************************************
IHELPS   NS
I                                        1   20HLINE              LINE NUMBER
I                                        4   9 HSCRN              'SCREEN'
I                                        5  27 HLN23              FIRST 23 CHARACTERS
I                                       18  25 HSCNAM             SCREEN NAME
I                                       28  82 HLN55              LAST 55 CHARACTERS
I/SPACE 2
I            DS
I                                        1  80 SLINE              S LINE
I                                        7  14 SNAME              FORMAT NAME
I/SPACE
I            DS
I                                        1  80 DLINE1             D LINE #1
I                                       19  200DLNNR              LINE NUMBER z
I                                       57  79 DPT1               TEXT LINE PART 1
I/SPACE
I            DS
I                                        1  80 DLINE2             D LINE #2
I                                        7  61 DPT2               TEXT LINE PART 2
I/SPACE
I            DS
I                                        1  80 FLINE              FIRST LINE IN FILE
I                                       24  29 ONAME              SOURCE NAME FOR COPY
I/SPACE
I            DS
I                                        1  80 LLINE              LAST LINE IN FILE
C/SPACE 2
C*   M A I N   L I N E   ***********************************************
C/SPACE
C*
C*  PERFORM HOUSEKEEPING FUNCTIONS
C*
C           HKFLG     IFEQ *BLANKS
C                     EXSR INIT
C                     END
C/SPACE
C*
C*  TEST FOR WORD SCREEN (INDICATES NEW OR FIRST LINE OF SCREEN)
C*
C/SPACE
C           CKFLG     IFEQ 'N'
C/SPACE
C           HSCRN     IFEQ 'Screen'
C                     MOVE 'Y'       SCFLG
C                     END
C/SPACE
C           HSCRN     IFEQ 'SCREEN'
C                     MOVE 'Y'       SCFLG
C                     END
C/SPACE
C                     END
C/SPACE
C           SCFLG     IFEQ 'Y'
C*
C*  PLACE FULL NAME FOR "S" SPECIFICATIONS AND FIRST 6 CHARACTERS
```

```
C*  FOR SOURCE NAME FOR TOLIBR
C*
C                   MOVE HSCNAM    SNAME
C*
C*  WRITE FIRST LINE OF FILE
C*
C           FLFLG     IFEQ 'N'
C                     MOVELHSCNAM    ONAME
C                     EXCPTFIRST
C                     MOVE 'Y'       FLFLG
C                     END
C*
C*  WRITE "S" SPECIFICATION
C*
C                     EXCPTASLINE                      - ASLINE
C                     MOVE 'Y'       CKFLG
C                     MOVE 'N'       SCFLG
C                     GOTO END
C                     END
C/SPACE
C*
C*  SPLIT EACH LINE INTO A "D" SPECIFICATION AND
C*  A CONTINUATION SPECIFICATION
C*
C           HLINE     IFGT 00
C           HLINE     IFLT 24
C           HLINE     ADD  1         DLNNR
C                     MOVE HLN23     DPT1
C                     MOVE HLN55     DPT2
C                     EXCPTADLIN1                      - ADLIN1
C                     EXCPTADLIN2                      - ADLIN2
C           HLINE     IFEQ 23
C                     MOVE 'N'       CKFLG
C                     END
C                     END
C                     END
C/SPACE
C           END       TAG                                                    END
C/SPACE 2
C*
C*  OUTPUT LAST LINE IN FILE FOR TOLIBR
C*
CLR                   EXCPTLAST
C*****************************************************************
C*  S U B R O U T I N E S                                         *
C*****************************************************************
C           INIT      BEGSR                                                  END
C*
C*  PREPARE OUTPUT LINES
C*
C                     MOVELSPEC,1    SLINE
C                     MOVELSPEC,2    DLINE1
C                     MOVELSPEC,3    DLINE2
C                     MOVELSPEC,4    FLINE
C                     MOVELSPEC,5    LLINE
C                     MOVE 'N'       HKFLG   1
C                     MOVE 'N'       FLFLG   1
C                     MOVE 'N'       SCFLG   1
C                     MOVE 'N'       CKFLG   1
C                     ENDSR                                                  END
O/SPACE 2
O*****************************************************************
O*  O U T P U T   S P E C I F I C A T I O N S                     *
O*****************************************************************
O/SPACE
OSDSPECS EADD             FIRST
O                         FLINE     80
OSDSPECS EADD             ASLINE
O                         SLINE     80
O/SPACE
OSDSPECS EADD             ADLIN1
O                         DLINE1    80
O/SPACE
OSDSPECS EADD             ADLIN2
O                         DLINE2    80
```

```
0158 O/SPACE
0159 OSDSPECS EADD              LAST
0160 O                          LLINE     80
**
     S          0124
     D          78  02Y                                   C                        X
     D
// COPY LIBRARY-S,NAME-
// CEND
```

Figure 20-17

S- and D-Specs

```
*    . . 1      ... 2  .      3  .      4         5         6         7 ..      8
// COPY LIBRARY-S,NAME-HLPTXT
     SHLPTXT01  0124
     D          780202Y                                   C                       X
     D HELP SCREEN CREATION DEMO SCREEN
     D          780302Y                                   C                       X
     D
     D          780402Y                                   C                       X
     D
     D          780502Y                                   C                       X
     D
     D          780602Y                                   C                       X
     D
     D          780702Y                                   C                    To X
     Dbe, or not to be: that is the question
     D          780802Y                                   C                    WheX
     Dther 'tis nobler in the mind to suffer
     D          780902Y                                   C                    TheX
     D slings and arrows of outrageous fortune,
     D          781002Y                                   C                    Or X
     Dto take arms against a sea of troubles,
     D          781102Y                                   C                    AndX
     D by opposing end them?  To die. to sleep,
     D          781202Y                                   C                    No X
     Dmore; and, by a sleep to say we end
     D          781302Y                                   C                    TheX
     D heart-ache and the thousand natural shocks
     D          781402Y                                   C                    ThaX
     Dt flesh is heir to, 'tis a consummation
     D          781502Y                                   C                    DevX
     Doutly to be wish'd  To die, to sleep,
     D          781602Y                                   C                    To X
     Dsleep. perchance to dream: ay, there's the rub;
     D          781702Y                                   C                       X
     D
     D          781802Y                                   C                       X
     D
     D          781902Y                                   C                       X
     D                          HAMLET Act 3 Scene 1
     D          782002Y                                   C                       X
     D                          William Shakespeare
     D          782102Y                                   C                       X
     D
     D          782202Y                                   C                       X
     D
     D          782302Y                                   C                       X
     D
     D          782402Y                                   C              PLEASE PREX
     DSS THE ENTER KEY TO RETURN TO PROCESSING
// CEND
```

Figure 20-18

Procedure HPMAKE

```
// IF DATAF1-SDSPECS DELETE SDSPECS,F1
// IF DATAF1-HELPS DELETE HELPS,F1
PCU VIRTDISK,DISKFILE,HELPS.SDF,PCV,,HELPS,CREATE,100,84
// LOAD HPMAKE
// FILE NAME-HELPS,LABEL-HELPS
// FILE NAME-SDSPECS,LABEL-SDSPECS,RECORDS-100
// RUN
TOLIBR SDSPECS,F1,,REPLACE,.....,LIBRARY
```

Figure 20-19
Sample .BAT file

```
STARTRTR
CFGVDSK
COPY HELPS.SDF F
STOPRTR
```

Customizing Screen Attributes in Menus

by Preston Sights

Code on diskette:
Procedure CRTMENU
Screen format member SAMPLE
Message member SAMPLE##

My first boss once explained to me, "First impressions are critical. The president of the company doesn't see the fancy programming technique that saves five lines of code — he sees the reports and screens. That is his perception of your work." The first thing anyone sees of your work design is the menus, so developing easy-to-read and aesthetically pleasing menus is an important part of your job as a programmer.

Unfortunately, the standard tool used to create menus, the Screen Design Aid (SDA) menu facility, does not support color or screen attributes such as highlighting, underlining, or reverse image for menus. These screen attributes can contribute greatly to the legibility and aesthetic appeal of a menu and, in fact, are used effectively on all of your application screens. So why exclude them from use in menus? You don't have to. And you need not go through SDA twice — once through the menu facility and once through the display format facility — to create attractive menus. By following a few simple rules of composition, you can bypass the restrictions of SDA's menu facility and create S/36 menus — in one pass — that have all the design flexibility of an application screen.

To create working menus without SDA's help, you need to understand the anatomy of a menu. A compiled menu consists of two different load members — a message load member and a screen format load member. The message load member stores the compiled procedure, command, or OCL statements to be invoked by the corresponding menu option numbers. The screen format load member contains the compiled S- and D-specs that define the screen's appearance. You use the CREATE procedure to compile the message load member from the message source member, and you use the FORMAT procedure to compile the screen format load member from the screen format source member.

You can begin menu creation by using SEU or POP to develop the message source member containing the statements that will be invoked by each menu option. These statements are written into a message source member in a predefined format (Figure 20-20). The first line of the message source member identifies the load member name to be created. This name must be the menu name followed by ##. The first line of the message member also specifies the maximum length of the text for each message (i.e., the length of the statements to be invoked by the menu). You should specify a 2, which indicates that the message text can be up to 225 characters long. However, because these messages are going to be interpreted by the command processor, you are actually limited to 120 characters (the limit allowed by the command processor) for each message.

Each subsequent line in the message member defines the procedure call, OCL statement, or operator command associated with each menu option. Each line consists of a four-digit menu item number (from 0001 to 0024) followed by a space and the statement text. This menu item number also acts as a MIC (message identification code) for retrieval of the command from the message member.

Note that the default record length of source members created by SEU is 96 bytes. If you expect to have statements longer than 96 bytes, you should specify a longer record length when saving the source member from the editor. If you have a longer statement that you want to continue onto the next source line, you should repeat the menu item number on the next line. (Figure 20-20 shows an example of this technique.) The total length for the statement text still must be 120 characters or less.

After you complete the message source member, you must create the screen format source member, which contains the S- and D-specs that define the layout of the screen. (Figure 20-21 shows a sample menu screen and Figure 20-22 shows the corresponding screen format source member.) The name of the screen format source member must be the same as the menu name.

You can code the S- and D-specs directly using a source editor such as SEU or FSEDIT, or you can use SDA to create the menu screen just as you would any application screen. The SSP requires that the screen have a few fields of predefined length and type in the following order:

• A two-byte OUTPUT ONLY field must be defined to be used by the SSP for the workstation ID.

• An INPUT/OUTPUT field of the same size and position as the command input line on standard system menus must be included. For the S/36, the command line is one 120-byte field starting in position 3 on line 22. The 120-byte input field on the S/36 menus should be specified with a "normal" attribute (hexadecimal 20 in position 2 on line 22) to work with PC Support/36. You should enable the dup key for this field on the S/36.

• Line 24 (on the S/36) or line 22 (on the S/34) must be left blank because the SSP writes time, date, and other messages in this location.

• A CONSTANT (output) field, specified as nondisplay if indicator 05 is on, must show the workstation inquiry status. If another job has been suspended by the use of the ATTN key, this field will become visible. The standard text for this field is "CMD1-Resume job."

In addition to the above fields, S/36 menus should specify null fill (Y in position 27 of the S-spec). They also should enable the Roll keys and Command key 3 (Y in positions 28 and 37 and 56C in positions 64 through 66). Deviating from these minimum coding requirements may cause unpredictable results when the menu is executed. The actual menu text that you create has no restrictions except that you can use only constant fields. You may use any screen attributes you like.

Once the source members for the message and screen format member have been created, you need to compile them to produce the load members required for menu execution. Procedure CRTMENU (Figure 20-23) prompts for the menu name and runs both the CREATE procedure and the FORMAT procedure with the correct parameters (menu and library names). Your menu is then available for execution or maintenance.

This simple technique allows you to create menus that will be easy to read and that will meet your existing application screen standards. Menus developed with this method will help you make that "first impression" a favorable one.

Figure 20-20

Message member SAMPLE##

```
SAMPLE##.2
*
*   S/36 Sample Menu Message Member
*
0001 PCTRAN
PARM1,PARM2,PARM3.PARM4,PARM5.PARM6.PARM6.PARM7,PARM8.PARM9.PARM10.PARM11.PARM12.PARM13,PA
0001 RM14,PARM15,PARM16.PARM17
0002 PCTRAN  RENAME
0003 PCTRAN  DELETE
0004 PCTRAN  TESTFILE
0008 PCTRAN  XLT36FIL
0009 PCTRAN  XLT36PRT
0010 PCTRAN  XLTPCFIL
0011 PCTRAN  XLTPCPRT
0013 PCTRAN  FILETOPC
0014 PCTRAN  LIBRTOPC
0015 PCTRAN  PRNTTOPC
0016 PCTRAN  FILEFRPC
0017 PCTRAN  LIBRFRPC
0018 PCTRAN  PRNTFRPC
0020 PCTRAN  EDITABLE
0021 PCTRAN  COMPILE
0024 OFF
```

Figure 20-21

Sample menu

```
Menu - SAMPLE          Emulator Transfer Utility S/36       Workstation ID - 00
                 COPYRIGHT (c) 1985,1986 Software Systems Inc

              PC Functions                          Transfer Functions
 1  Allocate New PC File                   13. S/36 File        -> PC File
 2  Rename PC File                         14  S/36 libr Member -> PC File
 3  Delete PC File                         15  S/36 Print Item  -> PC Print File
 4  Test for Existence of PC File          16  PC File          -> S/36 File
 5                                         17  PC File          -> S/36 Libr Member
 6                                         18. PC Print File    -> S/36 Print Item
 7                                         19

          Translation Functions                        Translation Tables
 8  S/36 File       -> PC File             20  Edit Translation Table
 9  S/36 Print Item -> PC Print File       21. Compile Text Translation File
10  PC File         -> S/36 File           22
11  PC Print File   -> S/36 Print Item     23
12                                         24  Sign-Off

               Press 'CMD' Then '1' to Return to Program on Hold
Enter Menu Item Number of Program to Execute
```

Figure 20-22

Screen format member SAMPLE

```
*.      1   .     2         3        . 4       . . 5        6         7         8
    SSAMPLE     124         Y        Y                          A
    D           6 1 2Y                                     CMenu -
    DMENU       6 1 9Y                Y                    CSAMPLE
    D          30 125Y                Y                    CEmulator Transfer UtiliX
    Dty S/36
    D          16 162Y                                     CWorkstation ID -
    DWSID       2 179Y                Y
    D          45 218Y                Y                    CCOPYRIGHT (c) 1985,1986X
    D Software Systems Inc
    D          33 4 5Y                      Y              C          PC Functions X
    D
    D          33 444Y                      Y              C       Transfer FunctioX
    Dns
    D          36 5 2Y                                     C1  Allocate New PC FileX
    D
    D          37 540Y                                     C13  S/36 File        ->X
    D PC File
    D          36 6 2Y                                     C2  Rename PC File      X
    D
    D          37 640Y                                     C14  S/36 Libr Member ->X
    D PC File
    D          36 7 2Y                                     C3  Delete PC File      X
    D
    D          38 740Y                                     C15  S/36 Print Item  ->X
    D PC Print File
    D          36 8 2Y                                     C4  Test for Existence oX
    Df PC File
    D          37 840Y                                     C16  PC File          ->X
    D S/36 File
    D           2 9 2Y                                     C5
    D          40 940Y                                     C17  PC File          ->X
    D S/36 Libr Member
    D           210 2Y                                     C6
    D          391040Y                                     C18  PC Print File    ->X
    D S/36 Print Item
    D           211 2Y                                     C7
    D           31140Y                                     C19
    D          3213 5Y                      Y              C      Translation FunctX
    Dions
    D          341444Y                      Y              C         Translation TaX
    Dbles
    D          3614 1Y                                     C 8  S/36 File       -> X
    DPC File
    D          381440Y                                     C20  Edit Translation TaX
```

```
Dble
D         3615 1Y
DPC Print File
D         381540Y
Dation File
D         3616 1Y
DS/36 File
D          31640Y
D         3817 1Y
DS/36 Print Item
D          31740Y
D          318 1Y
D         121840Y
D         7320 3Y                   05Y
D' Then '1' to Return to Program on Hold
D         4421 3Y
Dof Program to Execute
DINPUT    12022 3   Y       Y    Y
```

```
C 9  S/36 Print Item -> X

C21. Compile Text TranslX

C10  PC File        -> X

C22.
C11. PC Print File   -> X

C23
C12
C24  Sign-Off
C            Press 'CMDX

CEnter Menu Item Number X
```

Figure 20-23

Procedure CRTMENU

```
*
*   CREATE MENU USING MESSAGE MEMBER AND SCREEN FORMAT AS SOURCE
*
// IF ?1?/  * 'ENTER THE MENU NAME TO BE COMPILED -'
// IF ?1R?/ CANCEL
// IF ?2?/  * 'DEFAULT LIBRARY OF ?2'?CLIB?'? USED'
*
CREATE /1?##,REPLACE,?2?
*
FORMAT CREATE,?1?,?2?,?1?,?2?,1,REPLACE,HALT,NOPRINT
```

Changing the Console Screen Format

answered by Matthew Henry and Jeff Pisarczyk

Code on diskette:
Procedures CPYFCPF, CPYFCPF2
Screen format member FCPF

Q We have a 3197 display station defined as a system console for three systems: one S/38 and two S/36s. The display station is connected locally to the S/38 and to the S/36s via IBM's Display Station Passthrough (DSPT). Our problem is identifying from which of the two S/36s we are operating; the system console screen does not provide a system name or other unique identifier. Displaying the physical location or serial number of each machine would save us time and effort. Is it possible to patch some SSP object to change the text SYSTEM CONSOLE at the top of the screen?

A To change the text at the top of the screen, you must change module ##FCPF in #LIBRARY, but be forewarned, we don't recommend changing the text because the system console display is in the same load module as the IPL screen. A mistake while changing the system console screen could lead to a system that cannot be IPLed. In addition, any changes you make to IBM screens are reset when you install a new SSP release or when you apply a load module PTF to load module ##FCPF. However, if you do decide to change the console display, follow these steps:

- Create a temporary library (called WORK01) to contain a working copy of load module ##FCPF.
- Use procedure CPYFCPF in Figure 20-24 to make two copies of the #LIBRARY load module ##FCPF in library WORK01. The first copy keeps the same name as the original and serves as your backup copy. The second copy is named ##FCPF2 and is the copy to which you apply the changes.
- Create a new source member (called FCPF) in library WORK01, and enter the $SFGR source statements exactly as shown in Figure 20-25; these statements define the original system console format.
- Change the value of the constant to a unique identifier in either line 2 or line 4 of source member FCPF. Do not change the length of the constants or the screen positions of any of the other information because you don't want to make any changes that would affect the input/output buffer positions.
- Apply the screen format changes in source member FCPF by using the FORMAT UPDATE procedure shown in Figure 20-26.
- You can verify that the changes to load module ##FCPF2 are correct by displaying the changed format with the SDA view option.
- After you verify the changes, use procedure CPYFCPF2 in Figure 20-27 to copy object member ##FCPF2 from library WORK01 to ##FCPF in library #LIBRARY.
- After dedicating your system, IPL the system to activate your changes.
- Save library WORK01 to diskette should load module ##FCPF be replaced in a new version of SSP or in a PTF.

Figure 20-24

Procedure CPYFCPF

```
// LOAD $MAINT
// RUN
// COPY TO-WORK01,FROM-#LIBRARY,NAME-##FCPF,LIBRARY-O
// COPY TO-WORK01,FROM-#LIBRARY,NAME-##FCPF,LIBRARY-O,NEWNAME-##FCPF2
// END
```

Figure 20-25

$SFGR source statements (FCPF) for console screen format

```
*.        1   .     2      .  3     .. 4         5         6         7         8
0001 S#$CPCON           06
0002 D        00100102Y                                   CSYSTEM
0003 D        00060159Y                Y  05              C   SUB
0004 D        00100166Y                Y                  CCONSOLE
0005 D        00020179Y
0006 D        0002210204
0007 D        0075210601
0008 D        0002220203
0009 D        0075220602
0010 D        00602306   Y0      Y
0011 D        00602406   Y0      Y
```

Figure 20-26

FORMAT UPDATE procedure

```
FORMAT UPDATE,##FCPF,#LIBRARY,FCPF,WORKO1,,,HALT,NOPRINT
```

Figure 20-27

Procedure CPYFCPF2

```
// LOAD $MAINT
// RUN
// COPY FROM-WORKO1,TO-#LIBRARY,LIBRARY-O,NAME-##FCPF2,NEWNAME-##FCPF,RETAIN-R
// END
```

Using 5250 Terminals in Data Mode

by Mel Beckman

Q Can you suggest a way to set up a 5251 workstation as an output-only monitor? I want to hang the displays from the walls in two rooms. One display will show useful, general-interest information, and the other will show priority information. A timer is necessary to update the displays periodically. I want to use the displays without attached keyboards, but I don't know how to sign on and start the job without a keyboard. I want to control the programs running at these displays from the system console, rather than initiate the programs from each workstation. This type of display is similar to the departure/arrival display at airports.

A Data Mode is an oft-ignored workstation mode that allows a program to acquire and work with the terminal without the user invoking the program. Contrast this mode with a command display station that requires a user to initiate jobs. You must configure Data Mode individually for each workstation that uses it by entering display type D. You can use a MRT-NEP program and $$TIMER to drive several displays with "airline tables" or similar output-only data. Or, you could use a separate single-requester terminal (SRT) program for each display station, using $$TIMER to delay between screen updates.

A $$TIMER value of five seconds would give adequate update frequency and put very little load on the system.

The sign-on requirement is a harder question. You still have to sign on at data-mode display stations, but only the user ID and password entry fields appear on the screen. After the sign-on, the data-mode display "waits" for a program to acquire it. If you want to minimize keying, you can make the user ID and password all X's (i.e., XXXXXXXX and XXXX). The user would have to press only the X key until auto-repeat filled the fields and then press ENTER to sign on. Your programs can start automatically at IPL and wait for their respective displays to become available before acquiring them.

Canceling Continuously Updating, Display-Only Programs

answered by Mel Beckman, Mike Patton, and Kenneth Sims

Q On our S/36, we run an interactive RPG II program that continuously displays an updated screen. Unfortunately, the only way the user can stop the job is by pressing the Attention key and selecting option 4 (set inquiry condition for program) via IBM's SUBR95. For performance reasons, we want to avoid using SUBR95, so we need to know whether it is possible, within the program, to intercept a key pressed by the user, similar to the interception you can perform using BASIC's INKEY$ instruction.

A Actually, a program that "interacts" with the user when the Attention key is pressed is not an interactive application but is a display application. However, try the following method to solve your problem: design a program that uses the $$TIMER function and specify a suitable period for timer expiration, such as 000001 (or 1 second). Output the screen, including any variable information, followed by an output of the $$TIMER format, which contains the time period for which the timer is to be set. Then read the workstation file. If the user has not pressed the Enter key or any of the command (function) keys, the *STATUS subfield in the INFDS informational data structure for the workstation will contain 1331, which signals timer expiration. If *STATUS does not contain 1331, your program will know that something (such as end-of-job) should occur. Otherwise, simply update the information to be displayed on the screen and redisplay the screen, followed by another output operation using the $$TIMER format. This cycle lets the user end the program by pressing the Enter key or any other command key you allow.

If you do not want to install ICF for $$TIMER, use SUBR95 as you have been doing. You are correct to avoid calling SUBR95 frequently, however, because it calls a system transient to check the flag.

Clearing the Last Screen Format When Using $$TIMER

by Darryn Lee

Many S/36 programmers are frustrated by the last screen format that remains on the screen when using $$TIMER. You can save the problem by using the OFF OCL command, but if you want to exit the program without signing off the system, the last format screen remains until you press ENTER.

After much work and searching, I solved the problem with only one line of code. The line of code must be added to the procedure between the // LOAD and // RUN statements; the source code does not change:

```
// LOAD XXX
.
.
.
// WORKSTN UNIT-?WS?,RESTORE-YES
.
.
.
// RUN
```

Diacritic Mode Explained

answered by Jeff Silden

Q I have a question about the use of the Grave Accent key. I know that the Command Key/Upper Shift and the Grave Accent key (the first key on the top row located to the right of the Cmd key) will reverse the display image, but the Command Key/Error Reset and the Grave Accent key put the display in a diacritic mode. I found this out when one of our users came to me because her screen was "funny." I got out of diacritic mode by powering off the display, but was curious when I could find no real explanation of the Grave Accent key. What are its uses?

A On the S/36, diacritic mode enables you to place a diacritic mark above or below a character to indicate a different phonetic value, often needed in languages other than English. You can enter only those diacritic marks that appear on one of the diacritic keys on the keyboard. The diacritics available for the 5251 include the ` (grave accent), the ´ (acute accent), the ¨ (diaresis), and the ç (cedilla).

To enter a diacritic above a character, enter diacratic mode, press the diacritic key, and then press the character. If the diacritic key and the character key form a valid combination, the cursor moves to the next position; if not, an error code (0029) is displayed.

The allowable characters for each diacritic mark follow: grave accent (A,E,I,O,U), acute accent (A,E,I,O,U), tilde (A,N,O), circumflex (A,E,I,O,U), diaresis (A,E,I,O,U, and y as a lowercase character only), and cedilla (C).

If only a diacritic mark is to be entered, press the diacritic mark and then the spacebar on the typewriter-like keyboard.

Diacritic mode is not directly enabled on the S/34. On the S/34, you need additional microcode specific to your geographical location. The S/36, on the other hand, is a "world class" computer. The support for diacritics is included in the workstation controller microcode. It is enabled by selecting the Multinational Character Set option during system installation/release update/microcode configuration. Diacritic mode applies only to the one character following the press of the grave accent. Thus, instead of powering off the terminal, you could have escaped the diacritic mode by simply pressing the Error Reset key or by pressing the space bar and then backspacing over the created diacritic. Last, just because you can display the diacritics, it doesn't mean you can print them.

Entering Special Characters on a Workstation

by Jeff Silden

Q On my 5362, I have found the hexadecimal function helpful when creating some of the special characters supported by the 3197 terminal that are also printed on the 3812 Pageprinter (e.g., fractions, slashed zeros, and so on), yet the hex key function on the 3197 isn't shown on the 5251 and 5291 terminal keyboards. I've also had trouble getting the hex key function to work on some of my clients' S/36s. Am I doing something wrong?

A As you noted, the hex key function lets you access more characters than those shown on your terminal keyboard. When using the 3180, 3196, and 3197 terminals, you select the hex function and then key a two-character value in the range of X'40' to X'FF' that corresponds to the desired hex code (values below X'40' are reserved for use by the workstation controller). You gain access to the hex key function on the 5251 and 5291 terminals by pressing the CMD key and then the Grave Accent key (to the left of the number 1 key) before keying the hex character you want.

But the terminal device is only half the story. The S/3X to which your terminal is connected must have a workstation controller that supports the hex functions. All AS/400s and S/36 models 5362, 5363, and 5364 support the hex key function, as do 5360s with the workstation expansion feature. 5360s that do not have the workstation expansion feature don't support the hex key function. It appears that the client machines with which you've experienced trouble are 5360s without the workstation expansion feature.

Differences Between 5251 and 5291 Character Sets

by Matthew Henry and Jeff Silden

Q Why on a 5251 terminal do the keystrokes Command Accent B7 produce the character 1/4, while the same combination of keystrokes produces a blank on the 5291 terminal?

A The 5291 does not display the same hex characters as a 5251. As a result, screens created on the 5251 with these special characters may appear differently on the 5291. To see what characters a terminal is capable of displaying, you can run the TESTREQ procedure from any command display. Select option 1 from the main menu, and then select option 2 to show all displayable characters for your terminal.

Toggling Cursor Sizes on 5291 and 5292 Workstations

answered by Mel Beckman

Q I've connected some IBM 5291/2 terminals to my S/36. The cursor is a rectangle like a one position zone in reverse image. It is really hard to tell where the cursor is when it stays on a one-position input field with reverse image attribute; the cursor is virtually invisible! How can I change the cursor to an underline cursor like on an IBM 5251 screen?

A To toggle the cursor between its rectangular block and underscore forms, press and hold down the Command key and press the Error Reset key. The 5292 Color Display Station can also display the cursor in either block or underline form. To choose or change its form, first enter the Select Option mode by pressing and holding the Command key and pressing the Error Reset key (a blue indicator will appear on the status line above the word Select). Next, press Numeric Key 1 and make your cursor choice. Numeric Key 2 in Select Option mode allows you to choose between blinking or nonblinking cursor action. After an option is selected or changed, press the Error Reset key to return to normal operation.

Fixing a 3197-D ROM Bug

answered by Mel Beckman

Q When we configure our 3197-D terminals with two sessions, using the "jump" key also shifts the Command key. Although the Caps Lock indicator is not on, pressing a Shift key resets the shifted Command key. Any solutions?

A Your problem is caused by a 3197 microcode bug. Your CE has a fix for this bug — new ROMs that contain relatively bug-free microcode — and because IBM feels the bug is the result of a manufacturing defect, the microcode is free even if your machine is out of warranty or not under maintenance.

Some CEs aren't very good at looking up problems on RETAIN or HONE — IBM's error-reporting and inquiry systems; I've had several CEs tell me there was no such fix even after I showed them the IBM letter to CEs announcing the fix! So be persistent, don't take "no" for an answer, and be sure that they upgrade all of your machines on site. You don't have to take your machines in for this fix even if you're under the carry-in maintenance plan because IBM does this upgrade on site.

Appendix A

S/36 Power Tools Program and Procedure Cross-Reference

Note: When copying certain programs from diskette, you will encounter the message

SYS-2594 Trying to copy privileged module.

Answer this message with option 0; however, if you are not signed on as a security officer, option 0 won't be available. In that case, take option 3, sign off and sign on again as a security officer, and copy the programs to disk.

Name	Member Type	Chapter	Page	Description
1PDS0$02	RPG program	15	500	Prints a sample letterhead to show IPDS functions. Called by LTHD1$00. Requires assembler subroutine SUBR50, SUBR51, and SUBR52 (IPDS Advanced Functions PRPQ).
#O	Screen format member	6	142	Screen format for menu #0.
#O##	Menu member	6	142	Menu that lets you return to your application library after quitting DisplayWrite/36. Called by TEXTDOC. Uses screen format member #0.
#QRYEXT	RPG code	7	166	F- and I-specs for file #QRYEXT.
#SCHED1	Procedure	13	375	Evokes daily job stream. Calls procedure #SCHED2.
#SCHED2	Procedure	13	375	Runs a daily job stream at a specific time of day. Called by #SCHED1.
@DATA	Assembler subroutine	3	66	Special device file that lets you use O- and I-specs to convert or format fields without performing disk I/O.
@DTE1	RPG subroutine	3	59	Converts Gregorian date to Julian date.
@DTE2	RPG subroutine	3	59	Converts Julian date to Gregorian date.
@DTLY	RPG subroutine	3	59	Determines if a year is a leap year.
@RPTSMPL	Procedure	15	470	Prints a sample report from an RPG program's O-specs. Called by REPTSMPL. Calls SMPLA@ . Generates an RPG program, SMPLB@, that actually prints the sample report.
ACTIVE	Assembler program	1	13	Returns the number of active jobs via the ?CD? substitution parameter.
AIBLD	RPG program	12	332	Reads disk VTOC and builds extract file containing parent and alternate index files. Called by AIUTIL.
AIDEL	RPG program	12	334	Passes the names of alternate files to the LDA for deletion. Called by AIUTIL.
AIDSP	RPG program	12	333	Prompts you to confirm reorganization of MAPICS files with alternate indexes. Called by AITUTIL. Uses screen format member AIDSPFM.
AIDSPFM	Screen format member	12	337	Screen format for program AIBSP.
AIUTIL	Procedure	12	330	Reorganizes MAPICS files that use alternate indexes. Calls programs AIBLD, AIDSP, and AIDEL.
ATRSET	Procedure	11	309	Sets library member attributes. Calls program ATRSET. Uses screen format member ATRSET.

Name	Member Type	Chapter	Page	Description
ATRSET	RPG program	11	310	Sets library member attributes. Called by ATRSET.
ATRSETFM	Screen format member	11	317	Prompt screen for procedure ATRSET.
BASUNL	Procedure	11	322	Unlocks a BASIC source program.
C24TO12	Procedure	3	55	Converts the system time from 24-hour to 12-hour format.
C24TO12A	RPG subroutine	3	52	Converts the system time from 24-hour to 12-hour format.
C24TO12B	RPG subroutine	3	53	Converts the system time from 24-hour to 12-hour format.
C24TO12C	RPG subroutine	3	54	Converts the system time from 24-hour to 12-hour format.
CADD	Procedure	18	660	Patches the CACHE program O#SVCMG to allow the CACHE procedure to be run from other than the system console.
CASDWMFM	Screen format member	6	125	Prompt screen for procedure CASDWM.
CASDWM	Procedure	6	125	Merges spool document with a DisplayWrite/36 document. Calls program CASDWM. Uses screen format member CASDWFM.
CASDWM	RPG program	6	126	Merges spool document with a DisplayWrite/36 document. Called by CASDWM.
CMPDAY	Procedure	3	62	Converts the day of week for the system date.
CMPDAY	RPG code	3	63	Converts the day of week for a given date.
COMPILE	Procedure	16	531	Copies procedure COMPILEC and gives it the same name as the program it compiles (preceded by $).
COMPILEC	Procedure	16	532	The RPG compile procedure copied and given the same name as the program it compiles (preceded by $). When RPGC runs, the compile listing will show the name of the program in the spool status display.
CPYFCPF	Procedure	20	720	Copies IBM load module ##FCPF to work library.
CPYFCPF2	Procedure	20	721	Copies IBM load module ##FCPF back to #LIBRARY.
CREM	Procedure	18	660	Patches the CACHE program O#SVCMG to disallow the CACHE procedure to be run from other than the system console.
CRMENU	Procedure	11	306	Re-creates a source member from a menu object member. Calls program CRSRC. Uses screen format member CRSRCFM.
CRMSG	Procedure	11	303	Re-creates a source member from a message object member. Calls program CRSRC. Uses screen format member CRSRCFM.
CRSRC	RPG program	11	305	Re-creates source from message and menu object members. Called by CRMENU and CRMSG.
CRSRCFM	Screen format member	11	304	Prompt screens for procedures CRMENU and CRMSG.
CRTEFL	Procedure	8	223	Creates an empty test file using the file specifications of the production file.
CRTMENU	Procedure	20	719	Creates a menu from a message member and screen format member.
CTRTXT	RPG code	3	66	Centers a line of text.
DEL	Procedure	8	242	Deletes multiple files.
DELMAP	Procedure	12	326	Deletes MAPICS backup diskettes.
DOGRP	Procedure	7	180	Prints vertical lines between structured opcodes and indents them. Calls program DOGRP. Uses screen format member DOGRPFM.
DOGRP	RPG program	7	180	Prints vertical lines between structured opcodes and indents them. Called by DOGRP.

Name	Member Type	Chapter	Page	Description
DOGRPFM	Screen format member	7	179	Prompt screen for procedure DOGRP.
DSP	Procedure	14	449	Browses a spool file with POP's file browse.
DUPCHR	RPG code	16	582	Performs character-by-character duplication in an input field when DUP key pressed.
DUPKEY	RPG program	16	585	Tests assembler subroutine SUBRDU. Called by DUPTST.
DUPTST	Procedure	16	584	Tests assembler subroutine SUBRDU. Calls program DUPKEY. Uses screen format member DUPTSTFM.
DUPTSTFM	Screen format member	16	584	Prompt screen for procedure DUPTST.
ERROR	Procedure	16	590	Displays error message and prompts for 0, 1, 2, or 3 option.
FCPF	Screen format member	20	720	Applies changes to the console screen format member.
FILEB	Procedure	14	448	Restricts POP's FILE display with a file mask. Copy to #LIBRARY.
FILEE	Procedure	14	449	Edits a file using Query Data Entry Facility. Copy to #POPLIB. Note that #POPLIB already contains a procedure named FILEE, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than E.
FILEKY6	Procedure	14	450	Switches to LIBR display. Copy to #POPLIB.
FILEL	Procedure	14	449	Links a file to its IDDU definition. Copy to #POPLIB.
FILEN	Procedure	14	433	Renames a single file in POP. Uses screen format member FILENFM. Copy to #POPLIB.
FILENM	Screen format member	14	434	Prompt screen for procedure FILEN. Copy to #POPLIB.
FILEQ	Procedure	14	437	Renames multiple files in POP. Calls procedures FQQ and FILVPARM. Uses screen format member FILEQQFM. Copy to #POPLIB. Note that another procedure is named FILEQ (FILEQ2 on diskette), so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.
FILEQ2	Procedure	14	450	Displays a file through Query/36 with IDDU field headers. Copy to #POPLIB and rename to FILEQ. Note that another procedure is named FILEQ, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.
FILEQQFM	Screen format member	14	436	Prompt screen for procedure FILEQ. Copy to #POPLIB.
FILES	Procedure	14	442	Copies multiple files in POP. Calls procedures FSQ and FILVPARM. Uses screen format member FILESSFM. Copy to #POPLIB.
FILESSFM	Screen format member	14	441	Prompt screen for procedure FILES. Copy to #POPLIB.
FILEU	Procedure	14	450	Unlinks a file from its IDDU definition. Copy to #POPLIB.
FILEZ	Procedure	14	446	Deletes a file without confirmation. Copy to #POPLIB.
FILVPARM	Procedure	14	445	Retrieves the next file name for renaming or copying. Called by FILEQ and FILES. Copy to #POPLIB.
FINDLAST	RPG code	8	214	Finds the last record number in a file.
FLDCMP	RPG program	13	364	Creates procedure to condense DisplayWrite/36 folders.
FLDCMPP	Procedure	13	364	Condenses DisplayWrite/36 folders.
FLEDIT	Procedure	8	201	Edits file records. Calls program FLEDIT.
FLEDIT	RPG program	8	195	Edits file records. Called by FLEDIT. Requires assembler subroutine SUBRFA.
FLEDITFM	Screen format member	8	200	Screen formats for program FLEDIT.

Name	Member Type	Chapter	Page	Description
FOLDMK	Procedure	9	251	Runs any SSP procedure on a group of selected folders. Calls program FOLDMK.
FOLDMK	RPG program	9	248	Builds a procedure to run an SSP procedure on selected folders. Called by FOLDMK. Requires assembler subroutine SUBRVR.
FOLDMKFM	Screen format member	9	250	Prompt screens for procedure FOLDMK.
FOLDMKMG	Message member	9	251	Message member for procedure FOLDMK.
FQQ	Procedure	14	440	Renames a file. Called from FILEQ. Copy to #POPLIB.
FSEDIT2S	Procedure	14	411	Allows two edit sessions in POP's FSEDIT. Insert this code into FSEDIT in #POPLIB as instructed in the article.
FSQ	Procedure	14	444	Copies a file. Called from FILES. Copy to #POPLIB.
FTSPRC	Procedure	2	26	Transmits or receives entire data files or library members to/from a remote S/36. Calls program FTSPRG. Uses screen format member FTSPRGFM.
FTSPRG	RPG program	2	27	Transmits or receives entire data files or library members to/from a remote S/36. Called by FTSPRC. Requires assembler subroutine SUBRF2 (part of IBM's Base Communications).
FTSPRGFM	Screen format member	2	28	Prompt screen for procedure FTSPRC.
GEOPK	Procedure	7	192	Saves a spooled document to a source member. Calls program GEOPK.
GEOPK	RPG program	7	192	Saves a spooled document to a source member. Called by GEOPK.
GOLEM	Assembler program	18	659	Grants console capability to any workstation. Called by GOLEM.
GOLEM	Procedure	18	659	Grants console capability to any workstation. Calls program GOLEM.
GTDTF1	COBOL program	16	570	Retrieves cursor position.
GTDTF2	COBOL program	16	571	Retrieves the DTF control block.
HISTCOPY	Procedure	8	243	Saves history file to file HIST.n.
HPMAKE	Procedure	20	714	Converts S/36 help screens created on a PC to S- and D-specs. Calls program HPMAKE.
HPMAKE	RPG program	20	711	Converts S/36 help screens created on a PC to S- and D-specs. Called by HPMAKE.
ICCALL	Procedure	16	563	Sample ICF-INTRA program that increments a passed parameter. Calls program ICCALL.
ICCALL	RPG program	16	562	Sample ICF-INTRA program that increments a passed parameter. Called by ICCALL.
ICMAIN	Procedure	16	562	Sample ICF-INTRA program that initiates a session, sends and receives data, and controls execution. Calls program ICMAIN.
ICMAIN	RPG program	16	560	Sample ICF-INTRA program that initiates a session, sends and receives data, and controls execution. Called by ICMAIN.
IDDUXL	Procedure	10	262	Creates RPG F- and I-specs from IDDU. Calls program IDDUXL. Uses screen format member IDDUXLPM.
IDDUXL	RPG program	10	263	Creates RPG F- and I-specs from IDDU. Called from procedure IDDUXL.
IDDUXLPM	Screen format member	10	262	Prompt for procedure IDDUXL.
J//	Procedure	18	652	Puts a single OCL statement on the job queue. This

Name	Member Type	Chapter	Page	Description
				procedure must have the program data attribute set. Calls procedure JOCL and program JOCL.
JOBQ02	RPG program	13	374	Adds a job to a list of jobs to be run at a given time. Called by JOBQ1.
JOBQ03	RPG program	13	375	Retrieves a job for execution from the list of jobs created by procedure JOBQ1. Called by JOBQ2.
JOBQ1	Procedure	13	373	Adds a job to a list of jobs to be run at a given time. Calls program JOBQ02.
JOBQ3	Procedure	13	374	Executes jobs in the list of jobs created by procedure JOBQ1. Calls program JOBQ03.
JOCL	Procedure	18	652	Executes the OCL command stored in the LDA. Called by J//.
JOCL	RPG program	18	652	Reads procedure command line and stores it in the LDA. Called by J//.
KEEPOPEN	Procedure	8	231	Keeps large indexed files open. Substitute your indexed file names and specifications in this procedure's FILE statements.
KPOPEN	RPG program	8	231	MRT program used to keep large indexed files open. Substitute your indexed file names and specifications in this program's F-specs.
LDA	Procedure	18	653	Displays and allows updating of the LDA and switches. Uses screen format member LDAFM.
LDAFM	Screen format member	18	654	Prompt screen for procedure LDA.
LETHDFIL	Source member	15	501	IPDS specifications for a sample letterhead. Used by procedure LTHD1$00.
LIB#2518	Message member	11	323	Message member for procedure LIB#DECR.
LIB#DECR	Procedure	11	323	Increases the size of #LIBRARY.
LIBBAK	Procedure	1	8	Saves one or all user libraries. Calls program LIBBAK. Uses screen format member LIBBAKFM and message member LIBMSG.
LIBBAK	RPG program	1	10	Creates a library backup procedure. Called by LIBBAK.
LIBBAKFM	Screen format member	1	7	Prompt screen for procedure LIBBAK.
LIBMSG	Message member	1	10	Message member for procedure LIBBAK.
LIBR*	Procedure	14	425	Placeholder used by LIBRO procedure. Copy to #POPLIB.
LIBRA	Procedure	14	450	Reallocates a library using ALOCLIBR. Copy to #POPLIB.
LIBRCOMP	Procedure	14	450	Condenses a library using the C opcode in LIBR. Insert at the beginning of POP's LIBRC procedure.
LIBRI	Procedure	14	406	Retrieves library and member information in POP. Calls program LIBRI. Copy to #POPLIB. Note that #POPLIB already contains a procedure named LIBRI, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than I.
LIBRI	RPG program	14	395	Retrieves library and member information in POP. Called by LIBRI. Uses screen format member LIBRIFM. Requires assembler subroutine SUBRLD. Copy to #POPLIB.
LIBRIFM	Screen format member	14	403	Screens for program LIBRI. Copy to #POPLIB.
LIBRKY8	Procedure	14	451	Switches to FILE display. Copy to #POPLIB.
LIBRL	Procedure	14	412	Emulates COBOLONL in POP, calling FSEDIT instead of SEU. Copy to #POPLIB.
LIBRM	Procedure	14	412	Removes diagnostics from RPG programs in POP. Copy to #POPLIB.

Name	Member Type	Chapter	Page	Description
LIBRO	Procedure	14	419	Transmits library members via ODF/36 and POP. Calls program ODFPOP and procedure SENDODF. Copy to #POPLIB. Note that another procedure is named LIBRO (LIBRO2 on diskette), so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than O.
LIBRO2	Procedure	14	412	Emulates RPGONL in POP, calling FSEDIT instead of SEU. Copy to #POPLIB and rename to LIBRO. Note that another procedure is named LIBRO, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than O.
LIBRQ	Procedure	14	413	Blanks out columns 1-5 and 75-80 in RPG source in POP. Calls program LIBRQ. Uses screen format member LIBRQFM. Copy to #POPLIB. Note that another procedure is named LIBRQ (LIBRQ2 on diskette), so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.
LIBRQ	RPG program	14	414	Blanks out columns 1-5 and 75-80 in RPG source in POP. Called by LIBRQ. Copy to #POPLIB.
LIBRQ2	Procedure	14	430	Puts a job on the job queue from POP. Copy to #POPLIB and rename to LIBRQ. Note that another procedure is named LIBRQ, so if you use both procedures in #POPLIB, one of them must be renamed to use a POP opcode other than Q.
LIBRQFM	Screen format member	14	415	Prompt screen for procedure LIBRQ. Copy to #POPLIB.
LIBRV	Procedure	14	431	Evokes a job from POP.
LOGIN	Procedure	17	605	Prevents you from signing on to more than one workstation. Calls procedure ONEUID. Use SECEDIT to make this the mandatory log-in procedure for each user. Change library name TOOLKIT to the library in which you store procedure ONEUID.
LOGO	Source member	15	497	IPDS specifications for a sample logo.
LONGPROC	Procedure	20	694	Demonstrates read under format technique for passing data from prompt screen to a multiscreen workstation program.
LOOPSAVE	Procedure	19	681	Executes an SSP backup command without rewinding the tape.
LTHD1$00	Procedure	19	500	Prints a sample letterhead to show IPDS functions. Calls program 1PDS0$02. Uses graphic source member LETHDFIL.
MAKE$F	Procedure	2	33	Generates a $FEFIX procedure to re-create a given O- or R-module. Calls program MAKE$F. Uses screen format member MAKE$F.
MAKE$F	RPG program	2	33	Generates a $FEFIX procedure to re-create a given O- or R-module. Called by MAKE$F. Requires assembler subroutine SUBRCS.
MAKE$FFM	Screen format member	2	32	Prompt screen for procedure MAKE$F.
MAKMEM	RPG program	2	38	Creates an empty $MAINT file with a directory entry. Called by an MKxxxxxx procedure (where xxxxxx is the name of the O- or R-module). MKxxxxxx procedures are created by procedure MAKE$F.
MAPLDA	Procedure	7	171	Prints a map of the LDA usage in a library and cross-reference reports of the LDA and RPG programs. Calls programs MPLD1, MPLD2, and MPLD3.

Name	Member Type	Chapter	Page	Description
MCOM	Procedure	4	75	Jobstream to edit menu command text and compile it (avoids having to use SDA to change command text).
MMETER	Procedure	13	390	Displays realtime memory usage. Calls program MMETER.
MMETER	RPG program	13	388	Displays realtime memory usage. Called by MMETER. Uses screen format member MMETERFM. Requires assembler subroutine SUBR$S.
MMETERFM	Screen format member	13	389	Screen format used by program MMETER.
MPLD1	RPG program	7	171	Prints a map of the LDA usage in a library. Called by MAPLDA.
MPLD2	RPG program	7	174	Prints LDA usage by field name. Called by MAPLDA.
MPLD3	RPG program	7	175	Prints LDA usage by field starting position. Called by MAPLDA.
MSG1404	Message member	15	487	Message member used by procedure PRINTS.
MSG1404N	Message member	15	487	Message member used by procedure PRINTS.
NEGLFT	RPG code	3	60	Adds a sign to negative numbers one position to the left of the leftmost digit.
NEST	Procedure	16	548	Prints action diagrams for structured verbs in an RPG program. Calls program NEST. Uses screen format member NESTFM.
NEST	RPG program	16	549	Prints action diagrams for structured verbs in an RPG program. Called by NEST.
NESTFM	Screen format member	16	548	Prompt screen for procedure NEST.
NEWDISK	Procedure	13	346	Automatically runs SMF at specified time.
NUMCK1	RPG code	3	57	Tests a field for all numeric values.
NUMCK2	RPG code	3	58	Tests a field for all numeric values.
ODFGET	RPG program	14	426	Sets up object transmission and prints reports. Called by SENDODF. Copy to #POPLIB.
ODFMSG	RPG code	14	425	F- and I-specs for file ODFMSG, which is used by programs ODFGET and ODFPOP.
ODFPOP	RPG program	14	420	Displays Send Objects Through the Network screen. Called by LIBRO. Uses screen format member ODFPOPFM. Copy to #POPLIB.
ODFPOPFM	Screen format member	14	423	Screen format for program ODFPOP. Copy to #POPLIB.
ODFSND	RPG code	14	425	F- and I-specs for file ODFSND, which is used by programs ODFGET and ODFPOP.
ONEUID	Procedure	17	606	Prevents user from signing on to more than one workstation. Called by ONEUID. Calls program ONEUID. Uses message member ONEUIDM.
ONEUID	RPG program	17	606	Tests to see whether user has signed on to more than one workstation.
ONEUIDM	Message member	17	608	Message member for procedure ONEUID.
PACKDATE	Procedure	16	580	Example of sorting a file containing packed dates.
POPLDA	Procedure	18	658	Retrieves saved LDA and UPSI switches from a stack. Calls program POPLDA.
POPLDA	RPG program	18	657	Retrieves saved LDA and UPSI switches from a stack. Called by POPLDA.
POS	Procedure	14	415	Positions the multimember LIBR display to a given member.
PRGLST	Procedure	11	284	Lists members created or modified within given date range. Calls program PRGLST.

Name	Member Type	Chapter	Page	Description
PRGLST	RPG program	11	285	Lists members created or modified within given date range. Called by PRGLST.
PRINT198	Procedure	15	483	Changes CPI after a report is created.
PRINTS	Procedure	15	487	Resets forms types for printing after IPL. Calls program PRINTS. Uses message members MSG1404 and MSG1404N.
PRINTS	RPG program	15	486	Resets forms types for printing after IPL. Called by PRINTS.
PROFL1	RPG program	16	525	Creates a new source member with profiling code. Note: the "source required" attribute must be set for the object member of this program. Called by PROFRPG.
PROFL2	RPG program	16	528	Sets number of elements in E-spec for # array (used to count statement executions). Called by PROFRPG.
PROFL3	RPG program	16	529	Prints source program listing merged with its profile data. Called by PROFPRT.
PROFPRT	Procedure	16	529	Prints source program listing merged with its profile data. Calls program PROFL3.
PROFRPG	Procedure	16	524	Inserts code into an RPG program to profile it. Calls programs PROFL1 and PROFL2.
PROMPT1	Procedure	20	694	Demonstrates read under format technique for passing data from prompt screen to a sort.
PROMPT	Procedure	20	695	Demonstrates read under format technique for passing data from prompt screen to a workstation program.
PROMRUF	Procedure	20	695	Demonstrates read under format technique for passing data from a workstation program to another.
PSHLDA	RPG program	18	656	Saves the LDA and UPSI switches on a stack. Called by PUSHLDA.
PUSHLDA	Procedure	18	658	Saves the LDA and UPSI switches on a stack. Calls program PSHLDA.
QQRYID	Procedure	10	256	Prints an enhanced query report header page. Calls program QQRYID.
QQRYID	RPG program	10	256	Prints an enhanced query report header page. Called by QQRYID.
QRYXRF	Procedure	7	164	Produces extract file of the queries contained in up to 8 libraries. Calls program QRYXRF. Uses screen format member QRYXRFFM. Note: change the value "foldername" in the last line of the procedure to the folder containing the IDDU definition for the extract file.
QRYXRF	RPG program	7	162	Produces extract file of the queries contained in up to 8 libraries. Called by QRYXRF. Uses assembler subroutines SUBRLD and SUBRLR.
QRYXRFFM	Screen format member	7	161	Prompt screen for procedure QRYXRF.
RANDOM	RPG subroutine	16	578	Generates a random number.
RBRIDGE	Assembler subroutine	16	569	Calls RPG assembler subroutine from COBOL. Used in program TBRIDG.
RECVDK	Procedure	5	89	Receives a diskette from another S/36 via BSC. Calls program RECVDK.
RECVDK	RPG program	5	87	Receives a diskette from another S/36 via BSC. Requires assembler subroutine SUBRDK. Called by RECVDK.

Name	Member Type	Chapter	Page	Description
REORG	Procedure	8	237	Generates a procedure (named REORG2) that deletes alternate indexes, reorganizes files, and rebuilds alternate indexes. Calls program REORG and procedure REORG2 (after it is generated).
REORG	RPG program	8	238	Generates a procedure that deletes alternate indexes, reorganizes files, and rebuilds alternate indexes. Called by REORG.
REORG1	Procedure	8	240	Reorganizes a file. Called by REORG1 (which is generated each time procedure REORG is run).
REPORT	Procedure	15	481	Prompts for report parameters. Uses screen format member REPORTFM.
REPORTFM	Screen format member	15	481	Prompt screen for procedure REPORT.
REPTSMPL	Screen format member	15	465	Prompt screen for procedure REPTSMPL.
REPTSMPL	Procedure	15	465	Accepts parameters to print a sample report from an RPG program's O-specs. Calls procedure @RPTSMPL. Uses screen format member REPTSMPL.
RESTFILE	Procedure	1	3	Restores a file to disk using a new name. Uses screen format member RESTFLFM.
RESTFLFM	Screen format member	1	5	Prompt screen for procedure RESTFILE.
ROLEM	Assembler program	18	659	Revokes console capability for a workstation. Called by ROLEM.
ROLEM	Procedure	18	658	Revokes console capability for a workstation. Calls program ROLEM.
RPGIN1	RPG program	16	537	Sample program to show traditional indicator use.
RPGIN3	RPG program	16	538	Sample program to show reduced indicator use.
RPGIN5	RPG program	16	540	Sample program to show better indicator use.
SAMPLE	Screen format member	20	718	Sample screen format used to show how to customize screen attributes in menus.
SAMPLE##	Message member	20	717	Sample message member used to show how to customize screen attributes in menus.
SENDDK	Procedure	5	88	Sends a diskette to another S/36 via BSC. Calls program SENDDK.
SENDDK	RPG program	5	85	Sends a diskette to another S/36 via BSC. Requires assembler subroutine SUBRDK. Called by SENDDK.
SENDODF	Procedure	14	426	Transmits library members via ODF/36. Called by LIBRO. Calls program ODFGET. Copy to #POPLIB.
SETDEL	Assembler program	8	242	Makes a file delete-capable. Called by SETDEL.
SETDEL	Procedure	8	241	Makes a file delete-capable. Calls program SETDEL.
SEUMOD	Procedure	4	72	Tests for existing member in #LIBRARY before saving it to the specified library. Uses screen format member SEUMODFM. Insert this code into IBM's #SEU procedure in #SEULIB library.
SEUMODFM	Screen format member	4	73	Prompt screen for procedure SEUMOD. Copy to library #SEULIB.
SFGRIO	Procedure	20	702	Creates externally described workstation files. Calls program SFGRIO. Uses screen format member SFGRIOFM and message member SFGRIOM1.
SFGRIO	RPG program	20	704	Creates externally described workstation files. Called by program SFGRIO.
SFGRIOFM	Screen format member	20	701	Prompt screen for procedure SFGRIO.

Name	Member Type	Chapter	Page	Description
SFGRIOM1	Message member	20	703	Message member for procedure SFGRIO.
SHOWUR	Procedure	8	213	Displays all locked records. Calls program SHOWUR.
SHOWUR	RPG program	8	210	Displays all locked records. Called by SHOWUR. Requires assembler subroutine SUBRUR.
SHOWURFM	Screen format member	8	212	Screen format for program SHOWUR.
SHRTAR	RPG program	15	466	Sample program used to show operation of procedure REPTSMPL.
SMFP21	Procedure	13	379	Analyzes SMF data and prints a cache analysis report. Calls programs SMFP21 and SMFP23.
SMFP21	RPG program	13	379	Extracts data from SMF.DATA file. Called by SMFP21.
SMFP23	RPG program	13	380	Prints a cache analysis report. Called by SMFP21.
SMPLA@	RPG program	15	471	Creates a program that prints a sample report from a program's O-specs.
SMPLD	RPG code	11	279	Sample code to read library directories. Requires assembler subroutine SUBRLD.
SMPSG	RPG code	11	288	Sample code to retrieve source and procedure members. Requires assembler subroutine SUBRSG.
SPACE	Procedure	5	104	Retrieves available space and volume ID for a diskette. Calls program SPACE.
SPACE	RPG program	5	105	Retrieves available space and volume ID for a diskette. Called by SPACE.
STKORG14	Procedure	8	235	Runs a dedicated COPYDATA for a file to remove deleted records. Substitute the file name and tailor the COPYDATA for your file.
STMBP01	Procedure	13	360	Improves disk compress by making room for work files.
STMBP02	Procedure	13	363	Increases file size and reorganizes file. Note: replace NEWSFILE with the name of your file, and adjust the value for parameter 11 to what's appropriate for your file.
SUBR##	Assembler subroutine	18	671	Retrieves the CPU serial number.
SUBR$F	Assembler subroutine	16	573	Searches for a string.
SUBR$S	Assembler subroutine	13	390	Special file that retrieves current memory usage information. Used in program MMETER.
SUBRAT	Assembler subroutine	3	69	Centers or left- or right-justifies a string.
SUBRCL	Assembler subroutine	15	455	Closes a printer file.
SUBRCP	Assembler subroutine	20	684	Retrieves cursor position.
SUBRCS	Assembler subroutine	2	40	Computes checksum. Used in program MAKE$F.
SUBRCS	Assembler subroutine	3	70	Converts a string between lowercase and uppercase.
SUBRDF	Assembler subroutine	18	672	Retrieves the system date format.
SUBRDK	Assembler subroutine	5	89	Reads and writes one or more diskette sectors. Used in programs RECVDK and SENDDK.
SUBRDP	Assembler subroutine	16	565	Turns on privileged mode.
SUBRDU	Assembler subroutine	16	585	Stores text in workstation's Dup key save area. Used in program DUPKEY.
SUBREK	Assembler subroutine	20	687	Enables function and command keys dynamically.
SUBRF5	Assembler subroutine	18	642	Retrieves Format-5s. Used in program VTOCFR.
SUBRFA	Assembler subroutine	8	203	Accesses files dynamically. Used in program FLEDIT.
SUBRFD	Assembler subroutine	18	667	Sends a message to the system console.

Name	Member Type	Chapter	Page	Description
SUBRLD	Assembler subroutine	11	280	Retrieves library directory information. Used in programs QRYXRF, LIBRI, and SMPLD.
SUBRLR	Assembler subroutine	7	168	Reads library member sectors. Used in program QRYXRF.
SUBRNP	Assembler subroutine	16	565	Turns off privileged mode.
SUBROP	Assembler subroutine	15	454	Opens a printer file.
SUBRRR	Assembler subroutine	20	685	Reads workstation input fields when Roll key pressed.
SUBRSG	Assembler subroutine	11	289	Retrieves source and procedure members from a library. Used in program SMPSG.
SUBRSW	Assembler subroutine	11	296	Writes source or procedure members to a library.
SUBRSX	Assembler subroutine	15	458	Retrieves the spool ID for a printer file. Used in program TESTSX.
SUBRSY	Assembler subroutine	18	668	Outputs to SYSLIST device.
SUBRUF	Assembler subroutine	8	207	Retrieves the user job information for each job accessing a given file. Used in programs TESTUF and TESTU.
SUBRUL	Assembler subroutine	11	274	Retrieves a library's users. Used in programs ONEUID, TESTU, and TESTUL.
SUBRUR	Assembler subroutine	8	213	Retrieves the RRN and user job information for each job accessing a given file. Used in program SHOWUR.
SUBRVR	Assembler subroutine	18	624	Retrieves VTOC entries. Used in programs FOLDMK and VGRAPH.
SYSERR	Procedure	18	670	Displays error message text for a given system error number.
SYSTM	Assembler program	18	674	Resets the system time without IPLing. Called by SYSTM.
SYSTM	Procedure	18	673	Resets the system time without IPLing. Calls program SYSTM.
TBRIDG	COBOL program	16	568	Tests assembler subroutine RBRIDG.
TESTSW	RPG program	11	294	Writes a test procedure into a library. Requires assembler subroutine SUBRSW.
TESTSX	Procedure	15	457	Tests assembler subroutine SUBRSX. Calls program TESTSX.
TESTSX	RPG program	15	457	Tests assembler subroutine SUBRSX. Called by TESTSX. Requires assembler subroutine SUBRSX.
TESTU	Procedure	18	645	Retrieves a file's or library's users. Calls program TESTU.
TESTU	RPG program	18	646	Retrieves a file's or library's users. Called by TESTU. Uses screen format member TESTUFM. Requires assembler subroutines SUBRUF and SUBRUL.
TESTUF	Procedure	8	207	Retrieves a file's users. Calls program TESTUF.
TESTUF	RPG program	8	207	Retrieves a file's users. Called by TESTUF. Requires assembler subroutine SUBRUF.
TESTUFM	Screen format member	18	645	Screen formats for program TESTU.
TESTUL	Procedure	11	273	Retrieves a library's users. Calls program TESTUL.
TESTUL	RPG program	11	274	Retrieves a library's users. Called by TESTUL.
TEXTDOC	Procedure	6	141	Transfers a user from their application library to #LIBRARY, automatically. Calls menu #0 (screen format #0 and menu member #0##). Copy this procedure to your user library (not #LIBRARY!).
TRYCMP	Procedure	11	291	Retrieves program source from a library. Calls program TRYCMP.

Name	Member Type	Chapter	Page	Description
TRYCMP	RPG program	11	291	Retrieves program source from a library. Called by TRYCMP.
UTERASE	Procedure	8	222	Erases all records in a file.
UTLIB1	RPG program	7	187	Builds empty TOLIBR file for copy into source and target libraries. Called by UTLLIB.
UTLIB2	RPG program	7	188	Compares two library directories. Called by UTLLIB.
UTLIB3	RPG program	7	189	Prints a report showing duplicate or outdated members in two libraries. Called by procedure UTLLIB.
UTLLIBFM	Screen format member	7	190	Prompt screen for procedure UTLLIB.
UTLLIB	Procedure	7	185	Prints a report showing duplicate or outdated members in two libraries. Calls programs UTLIB1, UTLIB2, and UTLIB3. Uses screen format member UTLIBFM.
VALDAY	Procedure	3	55	Validates the day portion of the date. Note that this procedure uses prompt member TDATE, format SCRN01, which you must provide.
VDSKTOA3	Procedure	13	362	Moves files to other disk spindles via tape.
VGRAPH	Procedure	18	623	Displays the VTOC graphically. Calls program VGRAPH.
VGRAPH	RPG program	18	614	Displays the VTOC graphically. Called by VGRAPH. Requires assembler subroutine SUBRVR.
VGRAPHFM	Screen format member	18	622	Screens used by program VGRAPH.
VTOCCM	Procedure	18	641	Compresses disks individually.
VTOCCM	RPG program	18	635	Prompt program for procedure VTOCCM. Uses screen format member VTOCFRFM.
VTOCFR	Procedure	18	640	Displays free disk space. Calls program VTOCFR and procedure VTOCCM.
VTOCFR	RPG program	18	628	Displays free disk space. Called by VTOCFR. Uses screen format member VTOCFRFM. Requires assembler subroutine SUBRF5.
VTOCFRFM	Screen format member	18	637	Screen formats for programs VTOCCM and VTOCFR.
WAITON	Procedure	16	587	Loops until a given procedure is found to be not running.
WHICH	Procedure	18	663	Improves on the DATAF1 Conditional Statement
WHICH	RPG program	18	664	Improves on the DATAF1 Conditional Statement
XREF	Procedure	7	149	Prints four cross-reference reports: File Label by Program, Program by File Label, Program by Procedure, and File Label by Procedure. Calls programs XREF01 through XREF05.
XREF01	RPG program	7	151	Creates extract file for cross-reference reports. Called by XREF.
XREF02	RPG program	7	155	Prints cross-reference report File Label by Program. Called by XREF.
XREF03	RPG program	7	156	Prints cross-reference report Program by File Label. Called by XREF.
XREF04	RPG program	7	157	Prints cross-reference report File Label by Procedure. Called by XREF.
XREF05	RPG program	7	158	Prints cross-reference report Program by Procedure. Called by XREF.

Index

Special Characters

A

B

C

D

H

I

J

L

M

O

P

Q

T