Smart DFSORT Tricks

DFSORT Web Site

For papers, online books, news, tips, examples and more, visit the DFSORT website at URL:

http://www.ibm.com/storage/dfsort

Contents

Smart DFSORT Tricks

Introduction: Details of functions used in tricks

For complete information on the DFSORT/ICETOOL functions used in the tricks shown here, see:

- "User Guide for DFSORT PTFs UK90007 and UK90006" at:

  http://www.ibm.com/servers/storage/support/software/sort/mvs/peug/

- "User Guide for DFSORT PTFs UQ95214 and UQ95213 at:

  http://www.ibm.com/servers/storage/support/software/sort/mvs/pdug/

- DFSORT documentation at:

  http://www.ibm.com/servers/storage/support/software/sort/mvs/srtmpub.html

Find and extract values from different positions

A customer asked us the following question:

> I am trying to get a list of copybooks used in a set of programs. I ran a search and my search result looks something like this.
>
> ```
> ASDF ASDFS COPY SDFSGWER
> ASDF    COPY SDFSGWSD
> ASDF        COPY SDFSGWAC
>    COPY SDFSGWSE
>   SDFSSDF         COPY SDFSGWSD
>       SDF     COPY SDFSGWAR
> ```
>
> The copybook names appear in different positions. I want to find each copybook name after 'COPY ' and extract it to the output file so I end up with just the list of copybook names like this:
>
> ```
> SDFSGWER
> SDFSGWSD
> SDFSGWAC
> SDFSGWSE
> SDFSGWSD
> SDFSGWAR
> ```
>
> Can DFSORT do this?

Using DFSORT's PARSE feature, we can find the 8-byte name after COPY and extract it into a %nn parsed fixed field. We can then build a new record with the %nn field.

Here's the DFSORT job that does the trick.

```
//S1     EXEC  PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SORTIN DD DSN=...  input file (FB/n)
//SORTOUT DD DSN=...  output file (FB/8)
//SYSIN    DD    *
  OPTION COPY
  INREC PARSE=(%=(ENDAT=C'COPY'),
              %00=(STARTAFT=BLANKS,FIXLEN=8)),
        BUILD=(%00)
/*
```

Here's how it works:

- ENDAT=C'COPY' finds the end of the 'COPY' string in each record. % is used because we don't need to extact anything at this point.

- STARTAFT=BLANKS finds the next non-blank character after 'COPY'. FIXLEN=8 extracts the 8 bytes starting at that non-blank into the %00 fixed parsed field.

- BUILD creates an output record with the 8-bytes we extracted into %00.

With the job above, if 'COPY' is not found in a record, the output record will contain blanks. If you don't want blank output records when the input doesn't have 'COPY', you can use DFSORT's substring search feature to only keep the records with 'COPY' in them:

```
  INCLUDE COND=(1,n,SS,EQ,C'COPY')
```

Extract and justify delimited fields

This question was posed on the IBMMAINFRAMES Help board:

> Does DFSORT support a variable layout with fields delimited by a separator? My input has fields of variable length delimited by semicolons like this:
>
> ```
> AAA;1;A2;13.2
> AAA;25;A;2.0
> AAA;3;A35;521.5
> AAA;4;A999;51.7
> ```
>
> I want to extract the delimited fields, right-justify the second and fourth fields, and left-justify the third field, so the output records look like this:
>
> ```
> AAA; 1;A2  ; 13.2
> AAA;25;A   ;  2.0
> AAA; 3;A35 ;521.3
> AAA; 4;A999; 51.7
> ```

Using DFSORT's PARSE features, we can extract the delimited fields into %nn parsed fixed fields. Using DFSORT's JFY feature, we can justify the extracted fields as needed. Here's the DFSORT job that does the trick:

```
//S1 EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SYMNAMES DD *
Fld1,1,4
Fld2,%00
Fld3,%01
Fld4,%02
Semi,';'
Blank,' '
//SORTIN DD DSN=...  input file
//SORTOUT DD DSN=...  output file
//SYSIN DD *
  OPTION COPY
* Extract second field into %00.
  INREC PARSE=(Fld2=(ABSPOS=5,FIXLEN=2,ENDBEFR=Semi),
* Extract third field into %01.
            Fld3=(FIXLEN=4,ENDBEFR=Semi),
* Extract fourth field into %02.
            Fld4=(FIXLEN=5,ENDBEFR=Blank)),
* Create output record with first field, semicolon,
* right-justified %00 field, semicolon,%01 field, semicolon
* and right-justified %02 field.
      BUILD=(Fld1,Fld2,JFY=(SHIFT=RIGHT),Semi,
            Fld3,Semi,Fld4,JFY=(SHIFT=RIGHT))
/*
```

We are using DFSORT Symbols for the fields and constants here as follows:

- Fld1: positions 1-4 - first field - fixed length

- Fld2: %00 - second field - variable length delimited by a semicolon

- Fld3: %01 - third field - variable length delimited by a semicolon

- Fld4: %02 - fourth field - variable length delimited by a blank

- Semi: constant for semicolon (';')

- Blank: constant for blank (' ')

You can use Symbols for fixed fields (p,m), parsed fields (%nn) and constants to make your DFSORT and ICETOOL control statements easier to code and understand.

Here's how the DFSORT job works:

- %00 extracts the 2-byte value that starts at position 5 and ends before the next semicolon. %00 extracts '1 ' for the first record, '25' for the second record, and so on.

- %01 extracts the 4-byte value that starts after the semicolon and ends before the next semicolon. %01 extracts 'A2 ' for the first record, 'A   ' for the second record, and so on.

- %02 extracts the 5-byte value that starts after the semicolon and ends before the next blank. %02 extracts '13.2 ' for the first record, '2.0 ' for the second record, and so on.

- BUILD creates an output record for each input record with input positions 1-4 ('AAA;'), the value in %00 right-justified (e.g. ' 1'), a semicolon, the value in %01 (e.g. 'A2 '), a semicolon, and the value in %02 right-justified (e.g. ' 13.2').

Squeeze out blanks or other characters

The following related questions were posed on the MVSHELP Help board:

- I have some input data like this:

```
Kevin           James           R
Douglas         Philips         K
Smith           John            L
```

where each of the three fields is 16 bytes. I want to remove all but one space between the fields so the output records look like this:

```
Kevin James R
Douglas Philips K
Smith John L
```

Can I use DFSORT to do this?

- I have some input data like this:

```
abcd!efghi!jklm!0!
abcdefgh!ijklmnopq!0!
```

I need to remove the ! characters and shift the remaining characters to the left so the output records look like this:

```
abcdefghijklm0
abcdefghijklmnopq0
```

Can I use DFSORT to do this?

Using DFSORT's SQZ function, we can "squeeze out" blanks or other characters. SQZ removes blanks automatically. MID=C' ' can be used to insert one blank for each group of blanks SQZ removes. PREBLANK=list can be used to remove one or more other characters as well.

These control statements would satisfy the first requirement:

```
  OPTION COPY
  INREC BUILD=(1,48,SQZ=(SHIFT=LEFT,MID=C' '))
```

These control statements would satisfy the second requirement:

```
  OPTION COPY
  INREC OVERLAY=(1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'!'))
```

Add leading and trailing apostrophes

This question was posed on the IBMMAINFRAMES Help board:

I have a sequential dataset with records as below:

```
ONE
TWO
THREE
FOUR
FIVE
```

I want to reformat the records for output as follows:

```
NUMBER 'ONE'
NUMBER 'TWO'
NUMBER 'THREE'
NUMBER 'FOUR'
NUMBER 'FIVE'
```

How can I do that?

You can do this quite easily with DFSORT's JFY function as follows:

```
OPTION COPY
INREC BUILD=(1,13,JFY=(SHIFT=LEFT,LEAD=C'NUMBER ''',
  TRAIL=C''''))
```

SHIFT=LEFT left-justifies the data. LEAD inserts NUMBER ' before the value with no intervening blanks.
TRAIL inserts ' after the value with no intervening blanks.

Deconstruct and reconstruct CSV records

This question was posed on the IBMMAINFRAMES Help board:

These are my input records:

```
"x@yz.e,q@yz.e","1,23",20,15
"_ansingh@sympatico.ca","56 131,44",0,1
```

I need to reverse the CSV fields to get these output records:

```
15,20,"1,23","x@yz.e,q@yz.e"
1,0,"56 131,44","_ansingh@sympatico.ca"
```

Can I use DFSORT to do this?

You can use DFSORT's PARSE function to extract the CSV fields into fixed parsed fields and then reformat them in reverse order with commas in between the fixed fields. Then you can use DFSORT's SQZ function to squeeze out the blanks to create the new CSV records. Note that if you just want to deconstruct the CSV fields, you can just use PARSE, and if you just want to construct CSV fields, you can just use SQZ.

Here's the DFSORT control statements that deconstruct and reconstruct the CSV records:

```
OPTION COPY
INREC PARSE=(%00=(ENDBEFR=C',',PAIR=QUOTE,FIXLEN=23),
             %01=(ENDBEFR=C',',PAIR=QUOTE,FIXLEN=11),
             %02=(ENDBEFR=C',',FIXLEN=2),
             %03=(FIXLEN=2)),
      BUILD=(%03,C',',%02,C',',%01,C',',%00)
OUTREC BUILD=(1,41,SQZ=(SHIFT=LEFT,PAIR=QUOTE),80:X)
```

Here's what the INREC statement does:

* %00 extracts the 23-byte value that starts at position 1 and ends before the next comma. PAIR=QUOTE ensures that a comma within the quoted string is not interpreted as the end of the CSV field.

* %01 extracts the 11-byte value that starts after the comma and ends before the next comma. PAIR=QUOTE ensures that a comma within the quoted string is not interpreted as the end of the CSV field.

* %02 extracts the 2-byte value that starts after the comma and ends before the next comma.

* %03 extracts the 2-byte value that starts after the comma.

- BULD creates an output record for each input record with the value in %03, a comma, the value in %02, a comma, the value in %01, a comma and the value in %00. At this point, the records would look like this:

```
15,20,"1,23"     ,"x@yz.e,q@yz.e"
1 ,0 ,"56 131,44","_ansingh@sympatico.ca"
```

The OUTREC statement uses SQZ to remove the blanks between the fields. PAIR=QUOTE ensures that blanks within quoted strings are not removed.

Only include records with today's date

A customer asked the following question:

Can I make use of DFSORT to sort all of the records from a VSAM file to a sequential file based on the current date? I have a C'yyyymmdd' date field starting at position 27 of each record. This job will be run every day. Each day I want only the records for that day to be copied into the sequential file from the VSAM file.

You can use INCLUDE and OMIT to select records using a variety of formats for today's date like C'yyyymmdd', C'yyyy/mm/dd', +yyyymmdd, C'yyyyddd', C'yyyy/ddd', +yyyyddd, C'yymmdd' and so on.

The DATE1 operand corresponds to a C'yyyymmdd' constant for today's date. So the following control statement will include only those records with a C'yyyymmdd' date in positions 27-34 equal to today's date:

```
INCLUDE COND=(27,8,CH,EQ,DATE1)
```

Some other examples: for date values in the form C'yyyy/mm/dd', you could use the DATE1(/) constant; for date values in the form C'yyyy-mm', you could use the DATE2(-) constant; for date values in the form P'yyyyddd', you could use the DATE3P constant; and for date values in the form Z'yymmdd' (2-digit year date), you could use the Y'DATE1' constant.

Of course, you can use the other comparison operators (NE, GT, GE, LT, LE) as well as EQ. For example, you could use GT to select records with dates after today, or LT to select records with dates before today.

Include records using relative dates

A customer asked the following question:

I have a VB file which has a date in column 14 in the format '2005-10-10'. I want only those records for which this date is greater than the current date - 56 days. Can DFSORT do this?

You can use INCLUDE and OMIT to select records using a variety of formats for past and future dates like C'yyyymmdd', C'yyyy/mm/dd', +yyyymmdd, C'yyyyddd', C'yyyy/ddd', +yyyyddd, C'yymmdd' and so on.

The DATE1(-)-56 operand corresponds to a C'yyyy-mm-dd' constant for today's date minus 56 days. So the following control statement will include only those records with a C'yyyy-mm-dd' date in positions 14-23 greater than today's date - 56 days.

```
INCLUDE COND=(14,10,CH,GT,DATE1(-)-56)
```

Some other examples: for 'yyyymm' + 3 months, you could use DATE2+3; for P'yyyyddd' - 150 days, you could use DATE3P-150; for Z'mmddyy' + 7 days, you could use Y'DATE1'+7.

Fields from different record types

DFSORT's IFTHEN clauses allow you to deal directly with fields from records in the same data set that have different layouts.

Consider a DCOLLECT data set. It has V-type records, D-type records, A-type records, and so on. Each has different fields in different places within the specific record type. Say you want to produce a report containing information from some fields in the V-type records and some fields in the D-type records. Since DFSORT requires that fields to be sorted or used in a report be in the same place in all of the records processed, the trick is to set up a DFSORT job to:

- Use an INCLUDE statement to keep only the V-type records and D-type records.

- Use an INREC statement with an IFTHEN clause to reformat the V-type records to contain the sort/report fields you need from the V-type records, and blank placeholders for the fields you need from the D-type records, and another IFTHEN clause to reformat the D-type records to contain the sort/report fields you need from the D-type records, and blank placeholders for the fields you need from the V-type records.

- Use a SORT statement against the reformatted V-type and D-type records to sort on the record type and other sort fields you set up previously, so that the records are arranged in the order you need for the report.

- Use an OUTFIL statement against the sorted records to print the report using the report fields you set up previously.

For DCOLLECT records, you must add 5 to the offset shown in the books for a field to use it as a position for DFSORT, due to differences in the way the starting position and RDW are documented.

Here's the DCOLLECT example:

```
//RUNIT JOB ...
//STEP1  EXEC  PGM=ICEMAN
//SYSOUT   DD SYSOUT=*
//SORTIN   DD DSN=YAEGER.DCOLLECT.TEST,DISP=SHR
//REPORT DD SYSOUT=*
//SYSIN    DD *           CONTROL STATEMENTS
***********************************************************
* GET NEEDED FIELDS INTO REFORMATTED V-TYPE (VOLUME)
* AND D-TYPE (DATA SET) RECORDS WITH FIELDS IN THE SAME
* POSITIONS, AS FOLLOWS:
*
* REFORMATTED V-TYPE RECORD
*
*   | VOLSER | 'V ' | BLANKS | VOLSER | VOLTYPE | BLANKS |
*       6      2      44        6        7         7
*
* REFORMATTED D-TYPE RECORD
*
*   | VOLSER | 'D ' | DSNAME | BLANKS | BLANKS  | CREDT  |
*       6      2      44        6        7         7
***********************************************************
```

```
** INCLUDE ONLY V-TYPE RECORDS AND D-TYPE RECORDS
   INCLUDE COND=(9,2,CH,EQ,C'V ',OR,9,2,CH,EQ,C'D ')
** CREATE REFORMATTED V-TYPE RECORDS
   INREC IFTHEN=(WHEN=(9,2,CH,EQ,C'V '),  - just V-type records
      BUILD=(1,4,       - RDW
             5:29,6,    - DCVVOLSR - volser (for sorting)
            11:9,2,     - DCURCTYP - record type (for sorting)
            13:44X,     - blank dsname (for report)
            57:29,6,    - DCVVOLSR - volser (for report)
            63:35,1,    - DCVFLAG1 - volume type flags
* Translate Volume type flag to Description for report,
* using lookup and change
                  CHANGE=(7,B'...1.....',C'PRIVATE',
                          B'...1....',C'PUBLIC',
                          B'....1...',C'STORAGE'),
            70:7X)),    - blank create date field (for report)
** CREATE REFORMATTED D-TYPE RECORDS
   IFTHEN=(WHEN=(9,2,CH,EQ,C'D '),  - just D-type records
      BUILD=(1,4,       - RDW
             5:83,6,    - DCDVOLSR - volser (for sorting)
            11:9,2,     - DCURCTYP - record type (for sorting)
            13:29,44,   - DCDDSNAM - dsname (for sorting/report)
            57:6X,      - blank volser field (for report)
            63:7X,      - blank volume type field (for report)
            70:109,4,PD,M11)) - DCDCREDT - create date (for report)
*************************************************************
* SORT ON VOLSER (ASCENDING), RECORD TYPE (DESCENDING -> V,D)
* AND DSNAME (ASCENDING)
*************************************************************
   SORT FORMAT=CH,
     FIELDS=(5,6,A,    - volser in ascending order
            11,2,D,    - record type in descending order (V, D)
            13,44,A)   - dsname in ascending order
*************************************************************
* PRINT REPORT USING FIELDS FROM REFORMATTED V-TYPE AND
* D-TYPE RECORDS
*************************************************************
   OUTFIL FNAMES=REPORT,VTOF,
     HEADER2=('Volume/Dataset Report',
       30:DATE=(MDY/),45:'Page ',PAGE=(M11,LENGTH=3),/,X,/,
       'Volume',10:'Type',20:'Creation Date',36:'Data Set',/,
       6C'-',10:7C'-',20:13C'-',36:44C'-'),
     OUTREC=(57,6,10:63,7,20:70,7,36:13,44)
/*
```

Notice that OUTFIL's lookup and change feature was used to change the flags in DCVFLAG1 to English descriptions (PRIVATE, PUBLIC, STORAGE), and OUTFIL's VTOF feature was used to produce an FBA report data set rather than a VBA report data set.

Here's an example of what a portion of the output from the report might look like. Notice that the volumes are in sorted order, and for each volume the fields from the V-record are followed by the fields from the D-records for that volume including the dsnames in sorted order.

```
Volume/Dataset Report        02/23/05      Page 001

Volume  Type      Creation Date   Data Set
------  -------   -------------   ------------------------------------
DFDSS0  PRIVATE
                  2004348         D225592.A
                  2004338         D70R.ACS.ROUTINES
                  2005002         D70R.APPL.INTF.LIB
                  2004338         D70R.TEMPFIX.LINKLIB
                  2004338         D70R.TOOLLIB
                  2005004         D70S.DSSTOOLS.LINKLIB
...


SYS002  STORAGE
                  2004276         DFPXA.ADEPT.OLD.TESTCASE
                  2005048         HSMACT.H3.BAKLOG.D96048.T044559
                  2005048         HSMACT.H3.CMDLOG.D96048.T044558
                  2005048         HSMACT.H3.DMPLOG.D96048.T044559
                  2005048         HSMACT.H3.MIGLOG.D96048.T044559
                  2005042         SYS1.VTOCIX.SYS002

...
```

Create files with matching and non-matching records

A customer asked us the following question:

I have two input files containing lists of names, as follows:

Input File1

```
Vicky
Frank
Carrie
Holly
David
```

Input File2

```
Karen
Holly
Carrie
Vicky
Mary
```

How can I create output files for the following:

- the names that appear in both File1 and File2

- the names that appear only in File1

- the names that appear only in File2

This case is simple because there are **no duplicates** within File1 or within File2. The more complex case where there are duplicates within File1 and within File2 is discussed later in this topic.

Below is an ICETOOL job that can do this match by using the SPLICE operator. SPLICE helps you perform various file join and match operations. The trick here is to add an identifier of '11' to the File1 records, add an

identifier of '22' to the File2 records, and splice the second id byte for matching records so we get '12' for names in both files, '11' for names only in File1 and '22' for names only in File2.

```
//S1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN1 DD DSN=...    input File1
//IN2 DD DSN=...    input File2
//T1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(TRK,(5,5))
//** USE MOD FOR T1
//  DISP=(MOD,PASS)
//OUT12 DD SYSOUT=*  names in File1 and File2
//OUT1 DD SYSOUT=*   names in File1 only
//OUT2 DD SYSOUT=*   names in File2 only
//TOOLIN DD *
* Add '11' identifier for FILE1 records.
COPY FROM(IN1) TO(T1) USING(CTL1)
* Add '22' identifier for FILE2 records.
COPY FROM(IN2) TO(T1) USING(CTL2)
* SPLICE to match up records and write them to their
* appropriate output files.
SPLICE FROM(T1) TO(OUT12) ON(1,10,CH) WITH(13,1) -
USING(CTL3) KEEPNODUPS
/*
//CTL1CNTL DD *
* Mark FILE1 records with '11'
  INREC OVERLAY=(12:C'11')
/*
//CTL2CNTL DD *
* Mark FILE2 records with '22'
  INREC OVERLAY=(12:C'22')
/*
//CTL3CNTL DD *
* Write matching records to OUT12 file. Remove id.
  OUTFIL FNAMES=OUT12,INCLUDE=(12,2,CH,EQ,C'12'),BUILD=(1,10)
* Write FILE1 only records to OUT1 file. Remove id.
  OUTFIL FNAMES=OUT1,INCLUDE=(12,2,CH,EQ,C'11'),BUILD=(1,10)
* Write FILE2 only records to OUT2 file. Remove id.
  OUTFIL FNAMES=OUT2,INCLUDE=(12,2,CH,EQ,C'22'),BUILD=(1,10)
/*
```

The output files will have the following records:

OUT12

```
Carrie
Holly
Vicky
```

OUT1

```
Frank
David
```

OUT2

```
Karen
Mary
```

You can use variations of the example above as appropriate. For example, consider this case:

Input File1

```
0001 Vicky        01
0015 Frank        02
0005 Carrie       01
0008 Holly        01
0101 David        02
```

Input File2

```
A Karen
D Holly
X Carrie
R Vicky
L Mary
```

The names appear in different columns in each file. We want two output files:

- the records from File1 that have a match in File2
- the records from File1 that don't have a match in File2

Here's a variation of the previous SPLICE example that handles this case:

```
//S2 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN1 DD DSN=&&I1,DISP=(OLD,PASS)
//IN2 DD DSN=&&I2,DISP=(OLD,PASS)
//T1 DD DSN=&&TQ,UNIT=SYSDA,SPACE=(TRK,(5,5)),
//** USE MOD FOR T2
// DISP=(MOD,PASS)
//OUT1 DD DSN=...  names in FILE1 with match in FILE2
//OUT2 DD DSN=...  names in FILE1 without match in FILE2
//TOOLIN DD *
* Add '11' identifier for FILE1 records.
COPY FROM(IN1) TO(T1) USING(CTL1)
* Put FILE2 name in same positions as FILE1 name.
* Add '22' identifier for FILE2 records.
COPY FROM(IN2) TO(T1) USING(CTL2)
* SPLICE to match up records and write FILE1 records
* to their appropriate output files.
SPLICE FROM(T1) TO(OUT1) ON(6,10,CH) WITH(22,1) -
  USING(CTL3) KEEPNODUPS
/*
//CTL1CNTL DD *
* Mark FILE1 records with '11'.
  INREC OVERLAY=(21:C'11')
/*
//CTL2CNTL DD *
* Move FILE1 name field.  Mark FILE2 records with '22'.
  INREC BUILD=(6:3,10,21:C'22')
/*
//CTL3CNTL DD *
* Write matching FILE1 records to OUT1 file.  Remove id.
  OUTFIL FNAMES=OUT1,INCLUDE=(21,2,CH,EQ,C'12'),BUILD=(1,20)
* Write non-matching FILE1 records to OUT2 file.  Remove id.
  OUTFIL FNAMES=OUT2,INCLUDE=(21,2,CH,EQ,C'11'),BUILD=(1,20)
/*
```

The output files will have the following records:

OUT1

```
0005 Carrie
0008 Holly
0001 Vicky
```

OUT2

```
0101 David
0015 Frank
```

The examples above do not have any duplicates within either of the input files. When the input files have **duplicates**, things get trickier, but we can still use DFSORT's ICETOOL to create the output files for the names in File1 and File2, the names only in File1 and the names only in File2. Let's say the two input files contain these names:

Input File1

```
Vicky    01
Vicky    02
David    01
Frank    01
Frank    02
Frank    03
Karen    01
Mary     01
Mary     02
```

Input File2

```
Vicky    03
Vicky    04
Holly    01
Carrie   01
Carrie   02
Frank    04
Karen    02
```

Below is an ICETOOL job that can do this more complex match using ICETOOL's SELECT and SPLICE opera-tors. First we add an identifier of 'DD' to one record for each name found in File1 **and** File2 (FIRSTDUP), add an identifier of 'UU' to one record for each name found only in File1 **or** only in File2 (NODUPS), add an identifier of '11' to the File1 records and add an identifier of '22' to the File2 records. Then we splice the second id byte for matching records so we get '1U' for names only in File1, '2U' for names only in File2, and an id other than '1U' or '2U' for names in both File and File2.

```
//S3 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN1 DD DSN=...    input File1
//IN2 DD DSN=...    input File2
//F1 DD DSN=&&F1,UNIT=SYSDA,SPACE=(CYL,(5,5)),
//** USE MOD FOR F1
// DISP=(MOD,PASS)
//T1 DD DSN=&&TX,UNIT=SYSDA,SPACE=(CYL,(5,5)),
//** USE MOD FOR T1
// DISP=(MOD,PASS)
//OUT12 DD SYSOUT=*  names in File1 and File2
//OUT1 DD SYSOUT=*   names in File1 only
//OUT2 DD SYSOUT=*   names in File2 only
//TOOLIN DD *
* Get first record with each name from FILE1.
  SELECT FROM(IN1) TO(F1) ON(1,10,CH) FIRST
* Get first record with each name from FILE2.
  SELECT FROM(IN2) TO(F1) ON(1,10,CH) FIRST
* Get one record with each name in FILE1 and FILE2
* and add 'DD' identifier.
  SELECT FROM(F1) TO(T1) ON(1,10,CH) FIRSTDUP USING(CTL1)
* Get one record with each name only in FILE1 or only in FILE2
* and add 'UU' identifier.
  SELECT FROM(F1) TO(T1) ON(1,10,CH) NODUPS USING(CTL2)
* Add '11' identifier for FILE1 records.
  COPY FROM(IN1) TO(T1) USING(CTL3)
* Add '22' identifier for FILE2 records.
  COPY FROM(IN2) TO(T1) USING(CTL4)
* SPLICE to match up records and write them to their
* appropriate output files.
  SPLICE FROM(T1) TO(OUT1) ON(1,10,CH) -
    WITHALL WITH(1,21) USING(CTL5)
/*
//CTL1CNTL DD *
* Mark records with FILE1/FILE2 match with 'DD'.
  OUTFIL FNAMES=T1,OVERLAY=(21:C'DD')
/*
//CTL2CNTL DD *
* Mark records without FILE1/FILE2 match with 'UU'.
  OUTFIL FNAMES=T1,OVERLAY=(21:C'UU')
/*
//CTL3CNTL DD *
* Mark FILE1 records with '11'.
  OUTFIL FNAMES=T1,OVERLAY=(21:C'11')
/*
//CTL4CNTL DD *
* Mark FILE2 records with '22'.
  OUTFIL FNAMES=T1,OVERLAY=(21:C'22')
/*
```

```
//CTL5CNTL DD *
* Write FILE1 only records to OUT1 file. Remove id.
  OUTFIL FNAMES=OUT1,INCLUDE=(21,2,CH,EQ,C'1U'),
    BUILD=(1,20)
* Write FILE2 only records to OUT2 file. Remove id.
  OUTFIL FNAMES=OUT2,INCLUDE=(21,2,CH,EQ,C'2U'),
    BUILD=(1,20)
* Write matching records to OUT12 file. Remove id.
  OUTFIL FNAMES=OUT12,SAVE,
    BUILD=(1,20)
/*
```

The output files will have the following records:

OUT12

```
Frank    01
Frank    02
Frank    03
Frank    04
Karen    01
Karen    02
Vicky    01
Vicky    02
Vicky    03
Vicky    04
```

OUT1

```
David    01
Mary     01
Mary     02
```

OUT2

```
Carrie   01
Carrie   02
Holly    01
```

Split a file to n output files dynamically

The following related questions were asked on the MVSFORUMS Help board:

- I am working on a request where we need to split the file into two halves. How do we do it dynamically?

- I want to split a file into equal parts, for example, split an input file to 6 output files, each with 1/6 of the records. Can it be done with DFSORT?

- I have an input file and I can't predict the number of records in it. It varies from 0 to 20000 records. I need to split this input file into 5 different output files. If the total number of records is not divided equally by 5, then I need the extra records in the last file.

Here's a DFSORT job that uses SPLIT1R dynamically to divide any number of input records among any number of output files and ensures that the records in each output file are contiguous. Just make the appropriate changes shown by <--- for the number of output files you want.

```
//S1     EXEC  PGM=ICETOOL
//TOOLMSG  DD  SYSOUT=*
//DFSMSG   DD  SYSOUT=*
//IN DD DSN=... input file
//T1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(TRK,(1,1)),DISP=(,PASS)
//C1 DD DSN=&&C1,UNIT=SYSDA,SPACE=(TRK,(1,1)),DISP=(,PASS)
//CTL3CNTL DD *
  OUTFIL FNAMES=(OUT01,OUT02,...,OUTnn),  <--- code OUT01-OUTnn
//    DD DSN=*.C1,VOL=REF=*.C1,DISP=(OLD,PASS)
//OUT01 DD DSN=... output file01
//OUT02 DD DSN=... output file02
...
//OUTnn DD DSN=... output filenn  <--- code OUT01-OUTnn
//TOOLIN DD *
* Get the record count.
COPY FROM(IN) USING(CTL1)
* Generate:
* SPLIT1R=x where x = count/nn.
* nn is the number of output files.
COPY FROM(T1) TO(C1) USING(CTL2)
* Use SPLIT1R=x to split records contiguously among
* the nn output files.
COPY FROM(IN) USING(CTL3)
/*
//CTL1CNTL DD *
  OUTFIL FNAMES=T1,REMOVECC,NODETAIL,
    TRAILER1=(COUNT=(M11,LENGTH=8))
/*
//CTL2CNTL DD *
  OUTREC BUILD=(2X,C'SPLIT1R=',
    1,8,ZD,DIV,+nn,                <--- set to nn
      TO=ZD,LENGTH=8,80:X)
/*
```

Five ways to split a data set

One of the most significant things OUTFIL can do is create multiple output data sets from one pass over an input data set.

Here are five ways you can use OUTFIL to split a data set into multiple parts. All five can be used with SORT, MERGE or COPY. For illustration, the examples shown here assume you want to split the data set into three output data sets, but you can actually split it into any number of output data sets.

Use SPLIT1R

SPLIT1R=n can be used to split an input data set into multiple output data sets each of which will have contiguous records. SPLIT1R=n writes n records to each output data set, and writes any extra records to the last output data set.

Here's an example of SPLIT1R=4 for an input data set with 14 records:

```
//SPLIT1R EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=Y897797.INPUT1,DISP=OLD
//OUT1 DD DSN=Y897797.SPLITR1,DISP=(NEW,CATLG),
//         SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT2 DD DSN=Y897797.SPLITR2,DISP=(NEW,CATLG),
//         SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT3 DD DSN=Y897797.SPLITR3,DISP=(NEW,CATLG),
//         SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD *
  SORT FIELDS=(21,5,FS,A)
  OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLIT1R=4
/*
```

The first four sorted records are written to the OUT1 data set, the second four sorted records are written to the
OUT2 data set, the third four sorted records are written to the OUT3 data set, and the remaining two records are
also written to the OUT3 data set.

The resulting output data sets would contain the following records:

Y897797.SPLITR1 (OUT1 DD)

```
  sorted record 1
  sorted record 2
  sorted record 3
  sorted record 4
```

Y897797.SPLITR2 (OUT2 DD)

```
  sorted record 5
  sorted record 6
  sorted record 7
  sorted record 8
```

Y897797.SPLITR3 (OUT3 DD)

```
  sorted record 9
  sorted record 10
  sorted record 11
  sorted record 12
  sorted record 13
  sorted record 14
```

Notice that the records in each output file are contiguous.

Use SPLIT

SPLIT is the easiest way to split an input data set into multiple output data sets if you don't need the records in
each output data set to be contiguous. SPLIT can be used to split the records as evenly as possible among the
output data sets. SPLIT writes one record to each output data set in rotation.

Here's an example of SPLIT for an input data set with 14 records:

```
//SPLIT EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=Y897797.INPUT1,DISP=OLD
//OUT1 DD DSN=Y897797.SPLIT1,DISP=(NEW,CATLG),
//        SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT2 DD DSN=Y897797.SPLIT2,DISP=(NEW,CATLG),
//        SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT3 DD DSN=Y897797.SPLIT3,DISP=(NEW,CATLG),
//        SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD *
  SORT FIELDS=(21,5,FS,A)
  OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLIT
/*
```

The first sorted record is written to the OUT1 data set, the second sorted record is written to the OUT2 data set, the third sorted record is written to the OUT3 data set, the fourth sorted record is written to the OUT1 data set, and so on in rotation.

The resulting output data sets would contain the following records:

Y897797.SPLIT1 (OUT1 DD)

```
  sorted record 1
  sorted record 4
  sorted record 7
  sorted record 10
  sorted record 13
```

Y897797.SPLIT2 (OUT2 DD)

```
  sorted record 2
  sorted record 5
  sorted record 8
  sorted record 11
  sorted record 14
```

Y897797.SPLIT3 (OUT3 DD)

```
  sorted record 3
  sorted record 6
  sorted record 9
  sorted record 12
```

Notice that the records in each output file are not contiguous.

Use SPLITBY

SPLITBY=n is another way to split the records evenly among the output data sets if you don't need the records in each output data set to be contiguous. It is similar to SPLIT, except that it writes n records to each output data set in rotation.

Here's an example of SPLITBY=n for an input data set with 53 records:

```
//SPLITBY EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=Y897797.IN1,DISP=OLD
//OUT1 DD DSN=Y897797.SPLITBY1,DISP=(NEW,CATLG),
//        SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT2 DD DSN=Y897797.SPLITBY2,DISP=(NEW,CATLG),
//        SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT3 DD DSN=Y897797.SPLITBY3,DISP=(NEW,CATLG),
//        SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD *
  SORT FIELDS=(21,5,FS,A)
  OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLITBY=10
/*
```

The first ten sorted records are written to the OUT1 data set, the second ten sorted records are written to the OUT2 data set, the third ten sorted records are written to the OUT3 data set, the fourth ten sorted record are written to the OUT1 data set, and so on in rotation.

The resulting output data sets would contain the following records:

```
Y897797.SPLITBY1 (OUT1 DD)

  sorted records 1-10
  sorted records 31-40

Y897797.SPLITBY2 (OUT2 DD)

  sorted records 11-20
  sorted records 41-50

Y897797.SPLITBY3 (OUT3 DD)

  sorted records 21-30
  sorted records 51-53
```

Notice that the records in each output file are not contiguous.

Use STARTREC and ENDREC

STARTREC=n and ENDREC=m can be used to select a sequential range of records to be included in each output data set. STARTREC=n starts processing at the nth record while ENDREC=m ends processing at the mth record.

Here's an example of STARTREC=n and ENDREC=m:

```
//RANGE EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=Y897797.INPUT2,DISP=OLD
//FRONT  DD DSN=Y897797.RANGE1,DISP=(NEW,CATLG),
//          SPACE=(CYL,(5,5)),UNIT=SYSDA
//MIDDLE DD DSN=Y897797.RANGE2,DISP=(NEW,CATLG),
//          SPACE=(CYL,(5,5)),UNIT=SYSDA
//BACK   DD DSN=Y897797.RANGE3,DISP=(NEW,CATLG),
//          SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD *
  OPTION COPY
  OUTFIL FNAMES=FRONT,ENDREC=500
  OUTFIL FNAMES=MIDDLE,STARTREC=501,ENDREC=2205
  OUTFIL FNAMES=BACK,STARTREC=2206
```

Input record 1 through input record 500 are written to the FRONT data set. Input record 501 through input record 2205 are written to the MIDDLE data set. Input record 2206 through the last input record are written to the BACK data set.

The resulting output data sets would contain the following records:

Y897797.RANGE1 (FRONT DD)

```
  input record 1
  input record 2
  ...
  input record 500
```

Y897797.RANGE2 (MIDDLE DD)

```
  input record 501
  input record 502
  ...
  input record 2205
```

Y897797.RANGE3 (BACK DD)

```
  input record 2206
  input record 2207
  ...
  last input record
```

Use INCLUDE, OMIT and SAVE

INCLUDE/OMIT and SAVE can be used to select specific records to be included in each output data set. The INCLUDE and OMIT operands provide all of the capabilities of the INCLUDE and OMIT statements including substring search and bit logic. SAVE can be used to select the records that are not selected for any other subset, eliminating the need to specify complex conditions.

Here's an example of INCLUDE and SAVE:

```
//SUBSET EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=Y897797.INPUT3,DISP=OLD
//OUT1 DD DSN=Y897797.SUBSET1,DISP=(NEW,CATLG),
//         SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT2 DD DSN=Y897797.SUBSET2,DISP=(NEW,CATLG),
//         SPACE=(CYL,(5,5)),UNIT=SYSDA
//OUT3 DD DSN=Y897797.SUBSET3,DISP=(NEW,CATLG),
//         SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD *
  OPTION COPY
  OUTFIL INCLUDE=(8,6,CH,EQ,C'ACCTNG'),FNAMES=OUT1
  OUTFIL INCLUDE=(8,6,CH,EQ,C'DVPMNT'),FNAMES=OUT2
  OUTFIL SAVE,FNAMES=OUT3
```

Records with ACCTNG in positions 8-13 are included in the OUT1 data set. Records with DVPMNT in positions 8-13 are included in the OUT2 data set. Records without ACCTNG or DVPMNT in positions 8-13 are written to the OUT3 data set.

So the resulting output data sets might contain the following records:

Y897797.SUBSET1 (OUT1 DD)

```
 J20   ACCTNG
 X52   ACCTNG
 ...
```

Y897797.SUBSET2 (OUT2 DD)

```
 P16   DVPMNT
 A51   DVPMNT
 ...
```

Y897797.SUBSET3 (OUT3 DD)

```
 R27   RESRCH
 Q51   ADMIN
 ...
```

Sum a number with a decimal point

This question was posed on the MVSHELP Help board:

> How can I sum a numeric field that has a sign and a decimal point. For example, if my input file has:
>
> ```
> ABC -000010.10
> ABC  000090.90
> ABC  000067.90
> QRS  000123.62
> QRS -000239.76
> ```
>
> I want my output file to have:
>
> ```
> ABC  000148.70
> QRS -000116.14
> ```

You can use DFSORT's SFF (signed free form) format to extract the sign and digits from numeric values in various forms, such as the sdddddd.dd values in this example, into ZD format and then convert the ZD values to other forms as needed. To get the sum (or total) of these sdddddd.dd values for each key, you can use SFF in an OUTFIL statement with SECTIONS and TRAILER3. Here's a DFSORT job to do that:

```
//DECSUM EXEC  PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SORTIN DD DSN=...    input file
//SORTOUT  DD DSN=...  output file
//SYSIN   DD    *
* Sort on key
 SORT FIELDS=(1,3,CH,A)
* Use SFF to get total of extacted digits and sign from
* sdddddd.dd numbers for each key, and convert the
* totals back to sdddddd.dd numbers.
 OUTFIL REMOVECC,NODETAIL,
  SECTIONS=(1,3,
   TRAILER3=(1,4, copy bytes before number
    5:TOT=(5,10,SFF,EDIT=(STTTTTT.TT),SIGNS=(,-)), total/convert
    15:15,66))  copy bytes after number
/*
```

Some points that need explanation:

- REMOVECC removes the ASA carriage control character generated when SECTIONS is used. Since we're not creating a report, we don't want the ASA carriage control character.

- NODETAIL eliminates the data records. We only want the trailer record generated for each key by TRAILER3.

- SECTIONS and TRAILER3 creates one trailer record for the data records with each key.

- TOT gives us the total. We use SFF to extract the sign and digits from the sdddddd.dd values into ZD values so they can be totalled. We then convert the summed ZD values back to sdddddd.dd values using EDIT and SIGNS.

Alternatively, you can use a SUM statement to get the totals. But since SUM cannot use the SFF format directly, you need to use an INREC statement with SFF to convert the sdddddd.dd values into ZD values. Then you can use a SUM statement to sum the ZD values, and an OUTREC statement to convert the ZD values back to sdddddd.dd values. In this case, you can use OVERLAY in INREC and OUTREC to convert the values in place without specifying the bytes before or after the values. Here's the DFSORT job to do that:

```
//DECSUMA EXEC  PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SORTIN DD DSN=...    input file
//SORTOUT  DD DSN=...  output file
//SYSIN   DD    *
* Use SFF to convert the sdddddd.dd numbers to ZD numbers.
 INREC OVERLAY=(5:5,10,SFF,TO=ZD,LENGTH=10)
* Sort on key
 SORT FIELDS=(1,3,CH,A)
* SUM on the ZD numbers.
 SUM FIELDS=(5,10,ZD)
* Use EDIT and SIGNS to convert the ZD sums back to
* sdddddd.dd numbers.
 OUTREC OVERLAY=(5:5,10,ZD,EDIT=(STTTTTT.TT),SIGNS=(,-))
/*
```

Check for a numeric string

A customer asked us the following question:

Is it possible to replace the following with a simple, numeric string check?

```
INCLUDE COND=(1,1,CH,GE,C'0',
              AND,1,1,CH,LE,C'9',
              AND,2,1,CH,GE,C'0',
              AND,2,1,CH,LE,C'9',
              AND,3,1,CH,GE,C'0',
              AND,3,1,CH,LE,C'9',
              AND,4,1,CH,GE,C'0',
              AND,4,1,CH,LE,C'9',
              AND,5,1,CH,GE,C'0',
              AND,5,1,CH,LE,C'9',
              AND,6,1,CH,GE,C'0',
              AND,6,1,CH,LE,C'9')
```

This rather involved INCLUDE statement keeps only those records that have a character 0-9 in each of positions 1-6. The idea is to keep records like these:

```
123456
000001
987654
003218
```

and discard records like these:

```
A23456
000XY2
0ABCDE
75218R
```

We can use DFSORT's INCLUDE and OMIT numeric and non-numeric test capability to accomplish the requested numeric string check. Here's the INCLUDE statement we need:

```
INCLUDE COND=(1,6,FS,EQ,NUM)
```

FS format does a character numeric test; each byte of the specified field is checked for '0' through '9' (X'F0'-X'F9'). EQ,NUM tests for numerics. NE,NUM tests for non-numerics.

You can also use:

- ZD format to do a zoned decimal numeric test; each non-sign byte is checked for '0' through '9' (X'F0'-X'F9'), and the sign byte is checked for X'F0'-X'F9, X'D0'-X'D9' or X'C0'-X'C9'. For example, if you wanted to omit records with non-numeric ZD values in positions 21-28, you could use this OMIT statement:

  ```
  OMIT COND=(21,8,ZD,NE,NUM)
  ```

- PD format to do a packed decimal numeric test; each digit is checked for 0-9 and the sign is checked for F, D or C. For example, if you wanted to to include records with numeric PD values in positions 11-15, you could use this OUTFIL statement:

  ```
  OUTFIL INCLUDE=(11,5,PD,EQ,NUM)
  ```

  Note: Numeric tests can also be used in the WHEN condition of an IFTHEN clause. For example:

  ```
  INREC IFTHEN=(WHEN=(15,3,FS,NE,NUM),OVERLAY=(15:C'000'))
  ```

Change a C sign to an F sign in PD values

A customer asked the following question:

> I have three 5-byte packed decimal fields starting in positions 1, 6 and 11. The positive values have a C sign (e.g X'123456789C'), but I need them to have an F sign. The negative values have a D sign which I don't want to change. Can DFSORT change the sign from C to F for the positive values?

DFSORT's TO=PDF feature makes it easy to change the C sign to an F sign in your PD values. In this case, we can use OVERLAY to change the signs in the three specified fields without changing anything else in each record. Here's the control statements we'd use:

```
  OPTION COPY
  INREC OVERLAY=(1:1,5,PD,TO=PDF,
                 6:6,5,PD,TO=PDF,
                11:11,5,PD,TO=PDF)
```

TO=PDF sets an F sign for each positive PD value regardless of whether it originally had an F sign or a C sign. TO=PDC sets a C sign for positive PD values. TO=ZDF sets an F sign for positive ZD values. TO=ZDC sets a C sign for positive ZD values. The D sign is kept for negative values in all cases.

Display the number of input or output records

This question was posed on the MVSHELP Help board:

> I have some millions of records in my input file and I want to know only the number of records present in that input file. Can DFSORT display just the number of input records in an output file?

Here's a simple DFSORT job that will write just the record count to SORTOUT as an 8-digit number:

```
//CTRCDS EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=...   input file
//SORTOUT DD DSN=...  output file
//SYSIN DD *
  OPTION COPY
  OUTFIL NODETAIL,REMOVECC,
    TRAILER1=(COUNT=(M11,LENGTH=8))
/*
```

NODETAIL ensures that the data records are **not** included in the SORTOUT file. REMOVECC ensures that a '1' for the ANSI eject character is **not** included in the SORTOUT record. TRAILER1 with COUNT writes one record with the count to the SORTOUT file. M11,LENGTH=8 formats the count as 8 digits with leading zeros. So if the input file had five thousand records, SORTOUT would have:

```
00005000
```

Other edit masks could be used to format the count differently, if appropriate. For example, M10,LENGTH=8 formats the count as 8 digits with blanks substituted for leading zeros. So if the input file had five thousand records, SORTOUT would have:

```
    5000
```

If you delete records (for example, with OMIT), SORTOUT will have the count of the records you keep for output.

If you wanted to delete records with 'SUI' in positions 21-23, but display the count of input records and output records in SORTOUT, you could use a DFSORT job like this:

```
//CTINOUT EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=...   input file
//SORTOUT DD DSN=...  output file
//SYSIN DD *
  OPTION COPY
  INREC IFTHEN=(WHEN=INIT,
          OVERLAY=(1:C'00000000')),
        IFTHEN=(WHEN=(21,3,CH,EQ,C'SUI'),
          OVERLAY=(1:C'00000001'))
  OUTFIL NODETAIL,REMOVECC,
    TRAILER1=('Input count: ',COUNT=(M11,LENGTH=8),
              ', Output count: ',
               TOT=(1,8,ZD,M11,LENGTH=8))
/*
```

SORTOUT would have one record that might look like this:

```
Input count: 00000011, Output count: 00000006
```

Display SMF, TOD and ETOD date and time in readable form

Jorg Berning asked the following question on the IBM-MAIN mailing list:

> I'm trying to convert the SMF14TME field to the format hh:mm:ss. This field contains the "time since midnight, in hundredths of a second that the record was moved into the SMF buffer." Can I do this with DFSORT/ICETOOL?

DFSORT provides special formats that allow you to convert SMF, TOD and ETOD values to readable format.

You can use the SMF date and time formats (DT1-3 and TM1-4) in INREC, OUTREC, OUTFIL statements, and in ICETOOL's DISPLAY and OCCUR operators to display the normally unreadable SMF date and time values in a wide range of recognizable ways. For example, DT3 automatically converts the SMF date value to a Z'yyyyddd' (=C'yyyyddd') value and TM1 automatically converts the SMF time value to a Z'hhmmss' (=C'hhmmss') value. These ZD values can be used as is, or further edited using the editing features of DFSORT or ICETOOL.

Likewise, you can use the TOD date and time formats (DC1-3 and TC1-4) and the ETOD date and time formats (DE1-3 and TE1-4) in INREC, OUTREC, OUTFIL, DISPLAY and OCCUR to display the normally unreadable TOD (STCK) and ETOD (STCKE) date and time values, respectively, in a wide range of recognizable ways. For example, DC1 automatically converts the TOD date value to a Z'yyyymmdd' (=C'yyyymmdd') value and TE4 automatically converts the ETOD time value to a Z'hhmmssxx' (=C'hhmmssxx') value. These ZD values can be used as is, or further edited using the editing features of DFSORT or ICETOOL.

Here's an ICETOOL job that produces the report with readable SMF date and time that Jorg was looking for:

```
//SMFRPT JOB ...
//STEP0010 EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*    ICETOOL messages
//DFSMSG   DD SYSOUT=*    DFSORT  messages
//RAWSMF   DD DSN=...     SMF data
//SMF#     DD DSN=&&TEMPV,SPACE=(CYL,(15,15)),UNIT=SYSDA
//SMF#REP  DD SYSOUT=*    Report
//TOOLIN   DD *           ICETOOL statements
COPY FROM(RAWSMF) TO(SMF#) USING(SMFI)
DISPLAY FROM(SMF#) LIST(SMF#REP) -
  TITLE('SMF Type-14 Records') DATE TIME PAGE -
  HEADER('Time') ON(7,4,TM1,E'99:99:99')   - C'hh:mm:ss'
  HEADER('Date') ON(11,4,DT3,E'9999-999')  - C'yyyy-ddd'
  HEADER('Sys') ON(15,4,CH) -
  HEADER('SMF#') ON(6,1,BI) -
  HEADER('Jobname') ON(19,8,CH) -
  HEADER('Datasetname') ON(69,44,CH) -
  BLANK
/*
//SMFICNTL DD *
  INCLUDE COND=(6,1,BI,EQ,14)  - Select SMF Type-14 records only
/*
```

Here's a sample of what the report might look like:

```
SMF TYPE-14 RECORDS       02/24/05       10:35:11        - 1 -

TIME      DATE      SYS    SMF#   JOBNAME   DATASETNAME
--------  --------  ----   ----   --------  --------------------
21:30:01  2005-052  SMEP    14    TMVSLFS   TMON.MVS30.TMVINST
21:30:07  2005-052  SMEP    14    TMONADMP  TMON.SS.LMKLOAD
22:07:00  2005-052  SMEP    14    TMVSLFS   TMON.MVS30.TMVINST
22:07:00  2005-052  SMEP    14    TMVSLFS   TMON.MVS30.TMVINST
22:07:06  2005-053  SMEP    14    TMONADMP  TMON.SS.LMKLOAD
00:00:05  2005-053  SMEP    14    TMONMVS   TMON.SS.LMKASET
00:00:05  2005-053  SMEP    14    TMONCICS  TMON.STRATS.LMKASET
00:00:23  2005-053  SMEP    14    ESS148XC  SYS.EOR.CNTL
...
```

Include or omit groups of records

A customer asked us the following question:

I have a file with RECFM=FBA and LRECL=133 that contains a number of reports (up to 100). Each report consists of a header with a report id in the form 'RPT.cccccc' starting in position 2 followed by the detail lines for that report. Each report can have a different number of detail lines. For example:

```
      1RPT.FRANK
       LINE 01 FOR FRANK REPORT
       LINE 02 FOR FRANK REPORT
       LINE 03 FOR FRANK REPORT
       ...
      1LINE 61 FOR FRANK REPORT
       LINE 62 FOR FRANK REPORT
       ...
      1RPT.VICKY
       LINE 01 FOR VICKY REPORT
       LINE 02 FOR VICKY REPORT
       LINE 03 FOR VICKY REPORT
       LINE 04 FOR VICKY REPORT
       ...
      1RPT.CARRIE
       LINE 01 FOR CARRIE REPORT
       LINE 02 FOR CARRIE REPORT
       ...
      1RPT.HOLLY
       LINE 01 FOR HOLLY REPORT
       LINE 02 FOR HOLLY REPORT
       LINE 03 FOR HOLLY REPORT
       ...
      1RPT.MARY
       LINE 01 FOR MARY REPORT
       LINE 02 FOR MARY REPORT
       LINE 03 FOR MARY REPORT
       LINE 04 FOR MARY REPORT
       ...
      1RPT.DAVID
       LINE 01 FOR DAVID REPORT
       LINE 02 FOR DAVID REPORT
       ...
```

I want to extract different reports to an output file at different times. For example, one time I might want to extract the RPT.FRANK, RPT.HOLLY and RPT.MARY reports, and another time I might want to extract the RPT.CARRIE and RPT.DAVID reports. Can I do this with DFSORT?

Using DFSORT's IFTHEN clauses and SPLICE operator, we can add group numbers to the records, and propagate the report id (RPT.cccccc) to each record of the group. Then we can use an INCLUDE condition specifying the report ids for the groups we want to keep, or an OMIT condition specifying the report ids for the groups we want to delete.

We make a copy of the 10-byte report id ('RPT.cccccc') at the end of the record in positions 134-143. We follow that by the 8-byte group number (00000001 for the first group, 00000002 for the second group,and so on) in positions 144-151 and the 8-byte temp. number in positions 152-159. The temp. number is just used to get the group number.

Here's the DFSORT job that does the trick.

```
//S1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN  DD DSN=...  input file
//T1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//OUT  DD DSN=...  output file
//TOOLIN   DD *
```

```
* Reformat records by group as follows - the tempnum is
* just used to get the group number (00000001, 00000002, etc).
*  RPT.rptida               |RPT.rptida|00000001|blanks |
*   detail                  |blanks....|00000001|tempnum|
*   detail                  |blanks....|00000001|tempnum|
*  ...
*  RPT.rptidb               |RPT.rptidb|00000002|blanks |
*   detail                  |blanks....|00000002|tempnum|
*   detail                  |blanks... |00000002|tempnum|
*  ...
   COPY FROM(IN) TO(T1) USING(CTL1)
* Use the group number to splice 'RPT.cccccc' from each
* group header record into the detail records for that
* group as follows:
*  RPT.rptida               |RPT.rptida|00000001|blanks |
*   detail                  |RPT.rptida|00000001|tempnum|
*   detail                  |RPT.rptida|00000001|tempnum|
*  ...
*  RPT.rptidb               |RPT.rptidb|00000002|blanks |
*   detail                  |RPT.rptidb|00000002|tempnum|
*   detail                  |RPT.rptidb|00000002|tempnum|
*  ...
* Then do an INCLUDE using RPT.rptida for the groups you want.
   SPLICE FROM(T1) TO(OUT) ON(144,8,ZD) -
     WITHALL WITH(1,133) KEEPBASE USING(CTL2)
/*
//CTL1CNTL DD *
   INREC IFTHEN=(WHEN=INIT,OVERLAY=(144:SEQNUM,8,ZD)),
         IFTHEN=(WHEN=(2,4,CH,EQ,C'RPT.'),
                 OVERLAY=(134:2,10,144:SEQNUM,8,ZD)),
         IFTHEN=(WHEN=NONE,
                 OVERLAY=(152:SEQNUM,8,ZD,
                          144:144,8,ZD,SUB,152,8,ZD,M11,LENGTH=8))
/*
//CTL2CNTL DD *
   OUTFIL FNAMES=OUT,
* Use an INCLUDE with 134,7,CH,EQ,C'RPT.cccccc' for the
* groups you want.
     INCLUDE=(134,10,CH,EQ,C'RPT.FRANK',OR,
              134,10,CH,EQ,C'RPT.HOLLY',OR,
              134,10,CH,EQ,C'RPT.MARY'),
* Remove the copy of the report id, the group number and
* the temp. number.
     BUILD=(1,133)
/*
```

The INCLUDE condition selects groups RPT.FRANK, RPT.HOLLY and RPT.MARY, so OUT looks like this:

```
1RPT.FRANK
 LINE 01 FOR FRANK REPORT
 LINE 02 FOR FRANK REPORT
 LINE 03 FOR FRANK REPORT
 ...
1LINE 61 FOR FRANK REPORT
 LINE 62 FOR FRANK REPORT
 ...
1RPT.HOLLY
 LINE 01 FOR HOLLY REPORT
 LINE 02 FOR HOLLY REPORT
 LINE 03 FOR HOLLY REPORT
 ...
1RPT.MARY
 LINE 01 FOR MARY REPORT
 LINE 02 FOR MARY REPORT
 LINE 03 FOR MARY REPORT
 LINE 04 FOR MARY REPORT
 ...
```

Sort groups of records

A customer asked us the following question:

> I have a file with RECFM=FB and LRECL=30 that contains groups of records each of which has a header record, detail records and a trailer record. Here's an example of the input file:

```
HDR  2005/04/10
001     23.05-
002   5213.75+
003    861.51+
004    753.90-
TRL        T862143
HDR  2005/04/09
001    282.15+
002      8.00-
003   1496.28+
TRL        T201576
HDR  2005/03/11
001    123.86+
002     98.07-
003     61.16+
TRL        T031893
HDR  2005/04/09
001    106.27-
002   2308.00+
003     96.72+
004    206.99-
005    208.82-
TRL        T201573
```

> The header record is identified by 'HDR' and has the date in 'yyyy/mm/dd' format. The trailer record is identified by 'TRL'. All the other records are detail records with an amount in dddd.dds format (d is 0-9 and s is + or -).

I want to sort the groups by the date (ascending) in the header records, and also sort the detail records within each group by the amount (descending). If multiple groups have the same date (like the 2005/04/09 groups above), I want to keep their relative order. So my output should look like this:

```
HDR  2005/03/11
001    123.86+
003     61.16+
002     98.07-
TRL        T031893
HDR  2005/04/09
003   1496.28+
001    282.15+
002      8.00-
TRL        T201576
HDR  2005/04/09
002   2308.00+
003     96.72+
001    106.27-
004    206.99-
005    208.82-
TRL        T201573
HDR  2005/04/10
002   5213.75+
003    861.51+
001     23.05-
004    753.90-
TRL        T862143
```

Can I use DFSORT or ICETOOL to do this?

Using DFSORT's IFTHEN clauses and SPLICE operator, we can add group numbers to the records, propagate the date to each record of the group and add a "code" of 'A' for header records, 'B' for detail records and 'C' for trailer records. Then we can sort by the date and group number to get the groups in the right order, and sort by "code" and amount to keep the header and trailer records for each group in the right place while rearranging the amounts within each group.

We make a copy of the 10-byte date ('yyyy/mm/dd') at the end of the record in positions 31-40. We follow that with the "code" ('A', 'B' or 'C') in position 41, the 8-byte group number (00000001 for the first group, 00000002 for the second group,and so on) in positions 42-49, and the 8-byte temp. number in positions 50-57. The temp. number is just used to get the group number.

We use DFSORT's signed free form (SFF) format to sort the dddd.dds values.

Here's the DFSORT job that does the trick.

```
//S1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN DD DSN=...  input file
//T1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//T2 DD DSN=&&T2,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//OUT  DD DSN=...  output file
//TOOLIN   DD *
* Reformat records by group as follows - the tempnum is
* just used to get the group number (00000001, 00000002, etc).
* HDR  yyyy/mm/dd         |yyyy/mm/dd|A|groupnum|tempnum|
* xxx   dddd.dds          |          |B|groupnum|tempnum|
* xxx   dddd.dds          |          |B|groupnum|tempnum|
```

```
* ...
* TRL  ...                  |          |C|groupnum|tempnum|
  COPY FROM(IN) TO(T1) USING(CTL1)
* Use the group number to splice 'yyyy/mm/dd' from each
* group header record into the detail and trailer records for
* that group as follows:
* HDR  yyyy/mm/dd           |yyyy/mm/dd|A|groupnum|tempnum|
* xxx    dddd.dds           |yyyy/mm/dd|B|groupnum|tempnum|
* xxx    dddd.dds           |yyyy/mm/dd|B|groupnum|tempnum|
* ...
* TRL  ...                  |yyyy/mm/dd|C|groupnum|tempnum|
  SPLICE FROM(T1) TO(T2) ON(42,8,ZD) -
    WITHALL WITH(1,30) WITH(41,1) KEEPBASE
* Sort on the following fields:
*  - yyyy/mm/dd date ascending (to get groups in date order)
*  - group number ascending (to handle groups with same date)
*  - code byte ascending (to get header, detail and trailer
*    in right order)
*  - amount descending. (to get amount within groups in order).
*    Use SFF format to handle the decimal point and
*    trailing sign in the amount.
  SORT FROM(T2) TO(OUT) USING(CTL2)
/*
//CTL1CNTL DD *
  INREC IFTHEN=(WHEN=INIT,OVERLAY=(42:SEQNUM,8,ZD)),
* Set up the header records with date, code 'A' and groupnum.
       IFTHEN=(WHEN=(1,3,CH,EQ,C'HDR'),
               OVERLAY=(31:6,10,41:C'A',42:SEQNUM,8,ZD)),
* Set up the detail and trailer records with date, code 'B'
* and groupnum.
       IFTHEN=(WHEN=(1,3,CH,NE,C'HDR'),
               OVERLAY=(41:C'B',50:SEQNUM,8,ZD,
                        42:42,8,ZD,SUB,50,8,ZD,M11,LENGTH=8),
               HIT=NEXT),
* Overlay the code of 'B' with 'C' for the trailer records.
       IFTHEN=(WHEN=(1,3,CH,EQ,C'TRL'),
               OVERLAY=(41:C'C'))
/*
//CTL2CNTL DD *
* Sort by date, group number, code and amount.
  SORT FIELDS=(31,10,CH,A,   - date
     42,8,ZD,A,               - group number
     41,1,CH,A,               - code
     6,9,SFF,D)               - amount
* Remove date, code, group number and temp. number.
  OUTREC FIELDS=(1,30)
/*
```

Set RC=12 or RC=4 if file is empty, has more than n records, etc

The following related questions have been asked by various customers:

- I need to check if a data set is empty or not. If it's empty I want to skip all other steps. Is there any way I can check for empty data sets?

- I would like to skip certain steps in my job if a file is empty. This would be easiest if a utility would generate a non-zero RC if the input file is empty.

- I have several datasets that I need to sort together. How can I terminate if the total count of records in these data sets is greater than 5000.

- I have a file that always has a header and trailer record and may or may not have data records. Is there any way to check if there are no data records in the file?

ICETOOL can easily satisfy all of these requests. You can use ICETOOL's COUNT operator to set a return code of 12 or 4 if a specified data set is **EMPTY, NOTEMPTY, HIGHER(n), LOWER(n), EQUAL(n) or NOTEQUAL(n)**, where n is a specified number of records (for example, 5000). This makes it easy to control the execution of downstream operators or steps using JCL facilities like IF or COND. If you use the **RC4** operand, ICETOOL sets RC=4; otherwise it sets RC=12.

For example, in the following ICETOOL job, the EMPTY operand of COUNT is used to stop STEP2 from being executed if the IN data set is empty. ICETOOL sets RC=12 if the IN data set is empty, or RC=0 if the IN data set is not empty. ICETOOL only reads one record to determine if the data set is empty or not empty, regardless of how many records there are in the data set.

```
//STEP1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//IN DD DSN=...
//TOOLIN DD *
* SET RC=12 IF THE 'IN' DATA SET IS EMPTY, OR
* SET RC=0 IF THE 'IN' DATA SET IS NOT EMPTY
 COUNT FROM(IN) EMPTY
/*
// IF STEP1.RC = 0 THEN
//*** STEP2 WILL RUN IF 'IN' IS NOT EMPTY
//*** STEP2 WILL NOT RUN IF 'IN' IS EMPTY
//STEP2 EXEC ...
...
// ENDIF
```

In this next example, the HIGHER(5000) operand of COUNT is used to skip a SORT operation if the count of records in three data sets is greater than 5000. ICETOOL sets RC=12 if the CONCAT data sets have a total record count greater than 5000, or a RC=0 if the total record count is less than or equal to 5000. MODE STOP (the default) tells ICETOOL not to execute the SORT operation if the COUNT operation sets RC=12.

```
//SRT1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//CONCAT DD DSN=...
//       DD DSN=...
//       DD DSN=...
//OUT    DD DSN=...
//TOOLIN DD *
* SORT THE 'CONCAT' DATA SETS ONLY IF THEY
* HAVE LE 5000 RECORDS
MODE STOP
COUNT FROM(CONCAT) HIGHER(5000)
SORT FROM(CONCAT) TO(OUT) USING(CTL1)
/*
//CTL1CNTL DD *
  SORT FIELDS=(25,8,BI,A)
/*
```

In this last example, the EMPTY operand of COUNT is used to stop S2 from being executed if the INPUT data set doesn't have any data records between the header and trailer. An OMIT statement is used with COUNT to delete the header and trailer record, leaving only the data records as a subset of the INPUT data set. ICETOOL sets RC=4 (because the RC4 operand is specified) if the subset of records is empty, or RC=0 if the subset of records is not empty.

```
//S2 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//INPUT DD DSN=...
//TOOLIN DD *
* SET RC=4 IF 'INPUT' ONLY HAS A HEADER AND TRAILER, OR
* SET RC=0 IF 'INPUT' HAS DATA RECORDS.
 COUNT FROM(INPUT) EMPTY USING(HDTL) RC4
/*
//HDTLCNTL DD *
  OMIT COND=(1,6,CH,EQ,C'HEADER',OR,1,7,CH,EQ,C'TRAILER')
/*
// IF S1.RC = 0 THEN
//*** S2 WILL RUN IF 'IN' IS NOT EMPTY
//*** S2 WILL NOT RUN IF 'IN' IS EMPTY
//S2 EXEC ...
...
// ENDIF
```

Delete all members of a PDS

A customer asked us the following question:

I'm looking for a way with IBM utilities in JCL to delete all the members of a PDS. I was thinking of:

- Step 1: Use IKJEFT01 with LISTDS and MEMBERS to get the member names.

- Step 2: Somehow use DFSORT to construct IDCAMS control statements of the form:

  ```
   DELETE 'dsn(member)'
  ```

  for all of the member names produced by Step 1.

- Run IDCAMS against the control statements produced by Step 2.

  Can I use DFSORT for Step 2?

Here's the solution and explanation we provided to the customer.

Yes, DFSORT can do it. Note that IDCAMS does not accept a control statement like this:

```
 DELETE 'dsn(ABC     )'
```

so we need to squeeze out the trailing blanks like this:

```
 DELETE 'dsn(ABC)'
```

Fortunately, DFSORT's SQZ function can do that quite easily. Here's a job that will do the trick. For the example, we're using USERID.TEST.PDS as the name of the PDS. Be sure to replace USERID.TEST.PDS in the LISTDS and Symbol statement below with the actual name of your PDS.

```
//DELMEMS JOB ...
//* Generate LISTDS output
//STEP1    EXEC PGM=IKJEFT01
//SYSTSPRT DD DSN=&&MBRS,DISP=(,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA,
//           RECFM=FB,LRECL=80
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSTSIN  DD   *
 LISTDS 'USERID.TEST.PDS' MEMBERS
/*
//* DFSORT step to remove unwanted records from LISTDS output
//* and create DELETE statements acceptable to IDCAMS
//* (that is, with no trailing blanks)
//SHOWIT  EXEC  PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SYMNAMES DD *
name,'USERID.TEST.PDS'
/*
//SORTIN DD DSN=&&MBRS,DISP=(OLD,PASS)
//SORTOUT DD DSN=&&C1,UNIT=SYSDA,SPACE=(TRK,(5,5)),DISP=(,PASS)
//SYSIN     DD     *
  OPTION SKIPREC=9
  OMIT COND=(1,1,CH,NE,C' ')
  SORT FIELDS=COPY
  OUTREC IFOUTLEN=80,
    IFTHEN=(WHEN=INIT,
      BUILD=(C' DELETE ''',name,C'(',3,8,C')''')),
    IFTHEN=(WHEN=INIT,
       OVERLAY=(9:9,72,SQZ=(SHIFT=LEFT)))
/*
//* IDCAMS step to process DELETE statements generated by DFSORT
//STEP3   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DSN=&&C1,DISP=(OLD,PASS)
```

Some points that need explanation:

- We're using name as a DFSORT Symbol for the actual name of the PDS. This makes it easy to change the name without having to retype it in each BUILD operand.

- SKIPREC gets rid of the unwanted LISTDS header records.

- OMIT gets rid of the unwanted LISTDS trailer records.

- The two IFTHEN clauses handle building the DELETE statement for each member. The first IFTHEN clause builds statements of the form:

```
 DELETE 'name(mbr      )'
```

where mbr is 8 characters padded on the right with blanks. The second IFTHEN clause uses DFSORT's SQZ feature to squeeze out any blanks between the member name and the ')'. This creates the appropriate DELETE statement for each member name length (1 to 8 characters) without trailing blanks:

```
 DELETE 'name(mbr)'
```

Note that this trick deletes only member names, not ALIAS names.

It's a good idea to compress the PDS after you delete the members.

For PDSs with lots of members, you can improve the performance of the IDCAMS step by deleting the members in reverse order. To do this, just change:

```
SORT FIELDS=COPY
```

in the SHOWIT step to:

```
SORT FIELDS=(3,8,CH,D)
```

to sort the members in descending order.

Keep dropped duplicate records (XSUM)

A customer asked us the following question:

> When duplicates are eliminated using DFSORT, is it possible to capture the dropped duplicate records in a separate file. I use SUM FIELDS=NONE to eliminate the duplicates.

With SUM FIELDS=NONE and EQUALS in effect, DFSORT eliminates "duplicate records" by writing the first record with each key to the SORTOUT data set and deleting subsequent records with each key. A competitive product offers an XSUM operand that allows the deleted duplicate records to be written to an XSORTSUM data set. While DFSORT does not support the XSUM operand, DFSORT does provide the equivalent function and a lot more with the SELECT operator of ICETOOL.

SELECT lets you put the records that **are** selected in the TO data set and the records that **are not** selected in the DISCARD data set. So an ICETOOL SELECT job to do the XSUM function might look like this:

```
//XSUM JOB ...
//DOIT    EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//IN   DD DSN=...    input data set
//OUT DD DSN=...   first record with each key
//SORTXSUM DD DSN=...  subsequent records with each key
//TOOLIN   DD *
  SELECT FROM(IN) TO(OUT) ON(1,3,CH) FIRST DISCARD(SORTXSUM)
/*
```

This will put the first occurrence of each ON field (sort key) in the OUT data set and the rest of the records in the SORTXSUM data set.

If IN contained the following records:

```
J03 RECORD 1
M72 RECORD 1
M72 RECORD 2
J03 RECORD 2
A52 RECORD 1
M72 RECORD 3
```

OUT would contain the following records:

```
A52 RECORD 1
J03 RECORD 1
M72 RECORD 1
```

and XSORTSUM would contain the following records:

```
J03 RECORD 2
M72 RECORD 2
M72 RECORD 3
```

But SELECT can do much more than that. Besides FIRST, it also lets you use ALLDUPS, NODUPS, HIGHER(x), LOWER(y), EQUAL(v), LAST, FIRSTDUP and LASTDUP. And you can use TO(outdd) alone, DISCARD(savedd) alone, or TO(outdd) and DISCARD(savedd) together, for any of these operands. So you can create data sets with just the selected records, just non-selected records, or with both the selected records and non-selected records, for all of these cases.

Here's a few more SELECT statements to show some of its capabilities:

```
* Put duplicates in DUPS and non-duplicates in NODUPS
  SELECT FROM(DATA) TO(DUPS) ON(5,8,CH) ALLDUPS DISCARD(NODUPS)
* Put records with 5 occurrences (of the key) in EQ5
  SELECT FROM(DATA) TO(EQ5) ON(5,8,CH) EQUAL(5)
* Put records with more than 3 occurrences (of the key) in GT3, and
* records with 3 or less occurrences in LE3.
  SELECT FROM(DATA) TO(GT3) ON(5,8,CH) HIGHER(3) DISCARD(LE3)
* Put records with 9 or more occurrences in OUT2.
  SELECT FROM(DATA) ON(5,8,CH) LOWER(9) DISCARD(OUT2)
* Put last of each set of duplicates in DUP1
  SELECT FROM(DATA) TO(DUP1) ON(5,8,CH) LASTDUP
```

Create DFSORT Symbols from COBOL COPYs

The cobdfsym REXX program shown below allows you to create DFSORT Symbols from a COBOL copybook. Here's how:

* Wrap the COBOL copybook in a dummy program and compile the program to get a compiled listing.

* Use the cobdfsym REXX program to process the compiled listing and create a DFSORT symbols data set.

* Use that DFSORT symbols data set in a SYMNAMES DD statement for your DFSORT or ICETOOL jobs.

When you upload the cobdfsym program, watch out for the following:

*   Make sure the cobdfsym statements were not converted to uppercase. The cobdfsym statements must be mixed case as shown below.

* Make sure the REXX operators (for example, || for concatenation) were not translated to other characters. REXX must be able to recognize the operators.

cobdfsym translates COBOL data types to DFSORT formats according to the table shown in "DFSORT Formats for COBOL Data Types" in Appendix C of *z/OS DFSORT Application Programming Guide*. The actual translations performed can be summarized as follows:

* 'Group' to CH

* 'Grp-VarLen' to CH

* 'Display' to CH

* 'Disp-Num' without 'LEADING' and without 'SEPARATE' to ZD

* 'Disp-Num' with 'LEADING' and without 'SEPARATE' to CLO

* 'Disp-Num' with 'SEPARATE' and without 'LEADING' to CST

* 'Disp-Num' with 'LEADING' and with 'SEPARATE' to FS

- 'Packed-Dec' to PD

- 'Binary' with S9 to FI

- 'Binary' without S9 to BI

- 'Comp-1' to FL

- 'Comp-2' to FL

- Anything else to CH

cobdfsym converts COBOL 88 values to DFSORT symbols as follows:

- 'literal' to 'literal' (for example, 'APPROVED'). When a COBOL statement sets more than one literal, cobdfsym generates a DFSORT Symbols statement that sets the symbol to the **first** literal.

- decimal number to decimal number (for example, 14). When a COBOL statement sets more than one decimal number, cobdfsym generates a DFSORT Symbols statement that sets the symbol to the **first** decimal number.

- SPACES to ' ' (blank).

- ZERO to 0 (decimal zero).

- LOW-VALUE to X'00' (binary zero).

If any of the translated symbols do not meet your needs, you can fine-tune them by editing the DFSORT Symbols data set produced by cobdfsym.

Here's an example of a job to do the COBOL compile and REXX processing. //SYSLIB must point to the library that contains your COBOL copybook. //SYSPROC must point to the library in which you've stored cobdfsym. //SYMNAMES must point to the output data set or member for the DFSORT Symbols data set to be produced by cobdfsym; this data set must have, or will be given, RECFM=FB and LRECL=80.

Be sure to specify **MAP** in the PARM field.

```
//****************************************************************
//*  GENERATE SYMBOLIC NAMES FOR DFSORT USING COBOL COPYBOOK
//****************************************************************
//COBCOMP   EXEC PGM=IGYCRCTL,
// PARM=('APOST,RENT,NOSEQ,MAP',
//       'BUF(20K),OPT(STD),TERM,LIB')
//SYSLIB   DD  DSN=CLIB,DISP=SHR  -- library with COBOL copybook
//SYSPRINT DD  DSN=&&COBLIST,DISP=(,PASS),UNIT=SYSDA,
// SPACE=(TRK,(10,10))
//SYSTERM  DD  SYSOUT=*
//SYSIN    DD  *
      IDENTIFICATION DIVISION.
      PROGRAM-ID.      DUMMYPGM.
      ENVIRONMENT      DIVISION.
      DATA             DIVISION.
      WORKING-STORAGE  SECTION.
         COPY DCLSYM01.   -- COBOL COPY statement
      PROCEDURE DIVISION.
         GOBACK.
//SYSLIN   DD  DUMMY
//SYSUT1    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//SYSUT2    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//SYSUT3    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//SYSUT4    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//SYSUT5    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//SYSUT6    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//SYSUT7    DD SPACE=(CYL,(10),,,ROUND),UNIT=SYSDA
//****************************************************************
//*    INTERPRET COBOL LISTING AND CREATE DFSORT SYMNAMES DATA SET
//****************************************************************
//SYMNAMES EXEC PGM=IKJEFT1A,
//         COND=(04,LT),
//         PARM='%COBDFSYM'
//SYSPROC  DD DSN=RLIB,DISP=SHR  -- library with cobdfsym REXX program
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD  DUMMY
//COBLIST  DD  DSN=&&COBLIST,DISP=(OLD,PASS)
//SYMNAMES DD DSN=DFSORT.SYMBOLS(DCLSYM01),DISP=SHR -- DFSORT symbols
```

As an example, if the DCLSYM01 copybook looked like this:

```
      01  PACKAGE-RECORD.
          05  PACKAGE-HEADER.
              10  PACKAGE-HEADER-1       PIC  X(13).
              10  FILLER                 PIC  X.
              10  PACKAGE-HEADER-2       PIC  X(01).
              10  FILLER                 PIC  X.
              10  PACKAGE-SEQUENCE       PIC  9(08) COMP-3.
          05  CUSTOMER-GROUP.
              10  CG-NAME                PIC  X(30).
              10  CG-COUNT               PIC  9(10) COMP-3.
              10  CG-DATE                PIC  9(06) COMP.
              10  CG-TIME                PIC  9(08) COMP.
              10  CG-TYPE                PIC  S9(02) COMP.
              10  CG-LIMIT               PIC  S9(07).
              10  CG-STATUS              PIC  X(08).
                  88  APPROVED            VALUE 'APPROVED'.
                  88  DENIED              VALUE 'DENIED  '.
```

```
             88  PENDING                     VALUE SPACES.
         10  CG-COUNTY-NO                 PIC  99.
             88  DUTCHESS                    VALUE 14.
             88  KINGS                       VALUE 24.
             88  NOCOUNTY                    VALUE ZERO.
```

the DFSORT Symbols created in DFSORT.SYMBOLS(DCLSYM01) would look like this:

```
PACKAGE-RECORD,1,84,CH
PACKAGE-HEADER,1,21,CH
PACKAGE-HEADER-1,1,13,CH
PACKAGE-HEADER-2,15,1,CH
PACKAGE-SEQUENCE,17,5,PD
CUSTOMER-GROUP,22,63,CH
CG-NAME,22,30,CH
CG-COUNT,52,6,PD
CG-DATE,58,4,BI
CG-TIME,62,4,BI
CG-TYPE,66,2,FI
CG-LIMIT,68,7,ZD
CG-STATUS,75,8,CH
APPROVED,'APPROVED'
DENIED,'DENIED  '
PENDING,' '
CG-COUNTY-NO,83,2,ZD
DUTCHESS,14
KINGS,24
NOCOUNTY,0
```

You could use these symbols in a DFSORT job by specifying:

```
//S1 EXEC PGM=ICEMAN
//SYMNAMES DD DSN=DFSORT.SYMBOLS(DCLSYM01),DISP=SHR
...
```

You could use these symbols in an ICETOOL job by specifying:

```
//S2 EXEC PGM=ICETOOL
//SYMNAMES DD DSN=DFSORT.SYMBOLS(DCLSYM01),DISP=SHR
...
```

Here's the cobdfsym REXX program:

```
/*REXX - COBDFSYM : Create DFSORT symbols from COBOL listing
*** Freeware courtesy of SEB IT Partner and IBM ***
trace r
*/
call read_coblist
call fix_duplicates
call put_symnames
exit
Put_symnames:
/* Write generated symbol definitions */
   do i = 1 to nf
     queue dnam.i','dval.i
     say   dnam.i','dval.i
   end
/* Write appended  symbol definitions */
   do i = 1 to na
```

```
       queue dapp.i
       say   dapp.i
     end
     queue ''
     'EXECIO * DISKW SYMNAMES (FINIS'
     return
Put_line:
/*   Analyze Data Division Map line       */
     parse var line linenr level dataname .
     parse var dataname dataname '.' .
     if dataname = 'FILLER'   then Return
     if level = 'PROGRAM-ID'  then Return
     if level = 88            then Do
       nf = nf + 1
       dnam.nf  = dataname
       dval.nf  = d88.linenr
       dlvl.nf  = lev
       Return
     end
     blk       = substr(line,64,4)
     if level = 1 then nf = 0
     hexoff    = substr(line,79,3) || substr(line,83,3)
     if hexoff = '      ' then hexoff = '000000'
     parse var line 92 asmdef datatyp .
     if datatyp = 'Group' | datatyp = 'Grp-VarLen'
         then parse var asmdef . 'CL' len
         else do
          len = left(asmdef,length(asmdef)-1)
          if right(asmdef,2) = '1H' then len = 2
          if right(asmdef,2) = '1F' then len = 4
          if right(asmdef,2) = '2F' then len = 8
         end

     select
       when datatyp = 'Group'        then typ = 'CH'
       when datatyp = 'Grp-VarLen'   then typ = 'CH'
       when datatyp = 'Display'      then typ = 'CH'
       when datatyp = 'Disp-Num'     then typ = 'ZD'
       when datatyp = 'Packed-Dec'   then typ = 'PD'
       when datatyp = 'Binary'       then typ = 'FI'
       when datatyp = 'Comp-1'       then typ = 'FL'
       when datatyp = 'Comp-2'       then typ = 'FL'
       otherwise                          typ = 'CH'
     end
     if typ = 'FI' then do
       if s9.linenr /= 'Y' then typ = 'BI'
     end
     else do
       if typ = 'ZD' then
         if sp.linenr = 'Y' then
           if ld.linenr = 'Y' then typ = 'FS'
           else typ = 'CST'
         else
           if ld.linenr = 'Y' then typ = 'CLO'
     end
     off = 1 + x2d(hexoff)
     nf = nf + 1
     dnam.nf  = dataname
```

```
      dval.nf  = off','len','typ
      dlvl.nf  = lev
Return
Read_COBLIST:
   l88 = 0
   lx = 0
   na  = 0
   'EXECIO * DISKR COBLIST (FINIS'
   parse pull line
   do until substr(line,2,16)  = '  LineID  PL SL '
      parse pull line
   end
/* Process program text lines */
   do until substr(line,2,16) /= '  LineID  PL SL '
      parse pull line
      do until left(line,1) = '1'
         call Check_Code_line
         parse pull line
      end
      parse pull line
   end
/* Skip lines */
   do until substr(line,2,18)  = 'LineID   Data Name'
      parse pull line
   end
/* Process Data Division Map lines */
   do until substr(line,2,18) /= 'LineID   Data Name'
      parse pull line
      do until left(line,1) = '1'
         call Put_line
         parse pull line
      end
      parse pull line
      parse pull line
   end
/* Skip rest  */
   do until queued() = 0
      parse pull line
   end
Return
Fix_Duplicates:
/* Append _n to any duplicate data names */
   nd = 0
   tdup. = ''
   Do i = 1 to nf
      nam = dnam.i
      parse var tdup.nam flag i1
      if flag     = '' then do
         tdup.nam = '0' i
         iterate
      end
      if flag     = '0' then do
         nd = nd + 1
         td1.nd = i1 i
         tdup.nam = '1' nd
         iterate
      end
      td1.i1 = td1.i1 i
```

```
   End
   Do id = 1 to nd
      parse var td1.id i tail
      n = 0
      Do while i /= ''
         n = n + 1
         dnam.i = dnam.i || '_' || n
         parse var tail i tail
      End
   End
Return
Check_code_line:
/* Analyze program text line , capture 88 VALUE clauses */
/* Capture S9, LEADING, SEPARATE parameters              */
/* Make append lines from *+ comments                    */
   parse var line 4 linenr 10 flag . 19 . 25 stmt 91
   if linenr = '' then return
   linenr = linenr + 0
   if left(stmt,2) = '*+' then do
     na = na + 1
     dapp.na = substr(stmt,3)
     return
   end
   if left(stmt,1) = '*'  then return
   if left(stmt,1) = '/'  then return
   if lastpos('.',stmt) = 0 then do
     parse pull line
     if left(line,1) = '1' then parse pull line
     if substr(line,2,16) = '  LineID  PL SL ' then parse pull line
     parse var line 4 x1 10 x2 . 19 . 25 stmt2 91
     stmt = stmt||stmt2
   end
   parse var stmt w1 .
   if w1 = '88' then do
    l88 = linenr
    if l88 /= 0 then do
      parse var stmt . 'VALUE' tail
      if tail /= 0 then do
         parse var tail value '.' .
         d88.l88 = strip(value)
         if left(d88.l88,6) = 'SPACES'
           then d88.l88 = ''' '''
         if left(d88.l88,4) = 'ZERO'
           then d88.l88 = '0'
         if left(d88.l88,9) = 'LOW-VALUE'
           then d88.l88 = 'X''00'''
         l88 = 0
      end
    end
    return
   end
   else do
    lx = linenr
    if lx /= 0 then do
     parse var stmt x1 x2 x3
     if pos(' S9',x3) /=0 then s9.lx = 'Y'
     if pos(' LEADING',x3) /=0 then ld.lx ='Y'
     if pos(' SEPARATE',x3) /=0 then sp.lx = 'Y'
```

```
      lx = 0
    end
  end
Return
```

Join fields from two files on a key

This question was posed on the MVSHELP Help board:

I have two files with a key in bytes 1-3 and data in bytes 5-9.

File A has the following records:

```
000 $$$$$
001 AAAAA
002 CCCCC
003 EEEEE
004 GGGGG
```

and File B has the following records:

```
001 BBBBB
003 DDDDD
004 FFFFF
005 HHHHH
```

I want to join the data fields for pairs of records with the same key to get the following output:

```
001 AAAAA BBBBB
003 EEEEE DDDDD
004 GGGGG FFFFF
```

Can I do that using DFSORT/ICETOOL?

Below is an ICETOOL job that can do it. The trick is to reformat the fields of the IN1 and IN2 files so you can join them with the SPLICE operator. SPLICE helps you perform various file join and match operations.

Note: For this example, the key and data fields are in the same locations in both files. But this same technique can be used if the key and/or data fields are in different locations in the files by reformatting the two files appropriately.

```
//DFSORT  EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//IN1    DD DSN=... file 1
//IN2    DD DSN=... file 2
//TMP1   DD DSN=&&TEMP1,DISP=(MOD,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA
//OUT    DD DSN=...  output file
//TOOLIN DD *
* For this example, the fields (p,m) are as follows:
*   IN1:  sort key - 1,3
*         f1fld    - 5,5
*   IN2:  sort key - 1,3
*         f2fld    - 5,5
*
* Reformat the IN1 data set so it can be spliced
  COPY FROM(IN1) TO(TMP1) USING(CPY1)

* Reformat the IN2 data set so it can be spliced
  COPY FROM(IN2) TO(TMP1) USING(CPY2)

* Splice records with matching IN1/IN2 keys
  SPLICE FROM(TMP1) TO(OUT) ON(1,3,CH) WITH(11,5)
/*
//CPY1CNTL DD *
*  Use OUTREC to create |key |f1fld |blank|
 OUTREC FIELDS=(1:1,3,5:5,5,11:5X)
/*
//CPY2CNTL DD *
*  Use OUTREC to create:|key |blank |f2fld|
 OUTREC FIELDS=(1:1,3,11:5,5)
/*
```

Join fields from two files record-by-record

A customer asked the following question:

> I want to merge two files laterally record by record. For example:
>
> If FILE1 contains:
>
> ```
> AAAAA
> BBBBB
> CCCCC
> ```
>
> and FILE2 contains:
>
> ```
> 11111
> 22222
> 33333
> ```
>
> then my output file should contain:
>
> ```
> AAAAA11111
> BBBBB22222
> CCCCC33333
> ```
>
> Can DFSORT do this?

Normally, we're asked how to join files on a key, but in this case there is no key; we just want to join the files record-by-record. Well, no problem, we'll just create a key we can use by adding a sequence number (1, 2, ...) to the records in file1 and a sequence number (1, 2, to the records in file2. Then we can join the fields from the 1 records, from the 2 records, and so on using the same technique we used for joining files on a key.

Below is an ICETOOL job that can do this join by using the SPLICE operator. SPLICE helps you perform various file join and match operations.

```
//S1     EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//IN1    DD DSN=... file1
//IN2    DD DSN=... file2
//TMP1   DD DSN=&&TEMP1,DISP=(MOD,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA
//OUT    DD DSN=... output file
//TOOLIN DD *
* Reformat the IN1 data set so it can be spliced
 COPY FROM(IN1) TO(TMP1) USING(CTL1)
* Reformat the IN2 data set so it can be spliced
 COPY FROM(IN2) TO(TMP1) USING(CTL2)
* Splice records with matching sequence numbers.
 SPLICE FROM(TMP1) TO(OUT) ON(11,8,PD) WITH(6,5) USING(CTL3)
/*
//CTL1CNTL DD *
* Use OUTREC to create: |f1fld|blank|seqnum|
  OUTREC FIELDS=(1:1,5,11:SEQNUM,8,PD)
/*
//CTL2CNTL DD *
* Use OUTREC to create: |blank|f2fld|seqnum|
  OUTREC FIELDS=(6:1,5,11:SEQNUM,8,PD)
/*
//CTL3CNTL DD *
* Use OUTFIL OUTREC to remove the sequence number
 OUTFIL FNAMES=OUT,OUTREC=(1,10)
/*
```

VB to FB conversion

DFSORT makes it easy to do VB to FB conversion and FB to VB conversion..

OUTFIL adds lots of tricks to your DFSORT toolkit, including the ability to convert a variable-length data set to a fixed-length data set while sorting, copying or merging. With the VLFILL=byte option of OUTFIL, you can even do the conversion easily when "short" records are present.

The VTOF or CONVERT and OUTREC operands of OUTFIL can be used to change variable-length (e.g. VB) input records to fixed-length (e.g. FB) output records. VTOF or CONVERT indicates that conversion is to be performed and OUTREC defines the reformatted records. All output data sets for which VTOF or CONVERT is used must have or will be given fixed-length record formats.

Here's an example of OUTFIL conversion:

```
  SORT FIELDS=(7,8,CH,A)
  OUTFIL FNAMES=FB1,VTOF,OUTREC=(5,76)
```

The FB output records for the FB1 data set will be 76 byte records containing positions 5-80 of the VB input records.

Only positions 5-80 of VB input records longer than 80 bytes will be used for the 76-byte FB output records.

But what about VB input records that are "shorter" than the 80 bytes needed to copy input positions 5-80 to the 76-byte FB output records? No problem. DFSORT automatically uses the VLFILL=C' ' option with VTOF or CONVERT to replace missing bytes in "short" OUTFIL OUTREC fields with blanks. So all of your short VB input records will be padded with blanks to create 76-byte FB output records.

If you want to select your own padding byte, just specify the VLFILL=byte option. For example, here's how you'd use an asterisk as the padding byte for the previous example:

```
SORT FIELDS=(7,8,CH,A)
OUTFIL FNAMES=FB1,VTOF,OUTREC=(5,76),VLFILL=C'*'
```

FB to VB conversion

DFSORT makes it easy to do FB to VB conversion and VB to FB conversion.

The FTOV operand of OUTFIL can be used to change fixed-length (e.g. FB) input records to variable-length (e.g. VB) output records. If FTOV is specified without OUTREC, the entire FB input record is used to build the VB output record. If FTOV is specified with OUTREC, the specified fields from the FB input record are used to build the VB output record. The VB output records will consist of a 4-byte RDW followed by the FB data. All output data sets for which FTOV is used must have or will be given variable-length record formats.

Here's an example of FB to VB conversion:

```
OUTFIL FNAMES=FBVB1,FTOV
OUTFIL FNAMES=FBVB2,FTOV,OUTREC=(1,10,C'=',21,10)
```

The VB output records for the FBVB1 data set will contain a 4-byte RDW followed by the FB input record.

The VB output records for the FBVB2 data set will contain a 4-byte RDW followed by the characters from input positions 1-10, an '=' character, and the characters from input positions 21-30.

All of the VB output records created with FTOV will consist of:

RDW + input fields

Since all of the input fields from the FB input records are the same, all of the VB output records will be the same length. But if you have trailing characters such as blanks, asterisks, binary zeros, etc, you can create true VB output records with different lengths by using the VLTRIM=byte option of OUTFIL. VLTRIM=byte can be used with FTOV to remove trailing bytes of the specified type from the end of the VB output records. Here are some examples:

```
OUTFIL FNAMES=FBVB3,FTOV,VLTRIM=C'*'
OUTFIL FNAMES=FBVB4,FTOV,VLTRIM=X'40'
OUTFIL FNAMES=FBVB5,FTOV,VLTRIM=X'00'
```

FBVB3 will contain VB output records without trailing asterisks. FBVB4 will contain VB output records without trailing blanks. FBVB5 will contain VB output records without trailing binary zeros.

To further illustrate how FTOV and VLTRIM=byte work, say you have the following 17-byte FB data records that you want to convert to VB data records:

```
123456***********
0003*************
ABCDEFGHIJ*****22
*****************
```

If you use:

```
  OUTFIL FTOV
```

the following VB output records will be written (4-byte RDW followed by data):

```
Length | Data
   21     123456***********
   21     0003*************
   21     ABCDEFGHIJ*****22
   21     *****************
```

but if you use:

```
  OUTFIL FTOV,VLTRIM=C'*'
```

the following VB output records will be written (4-byte RDW followed by data):

```
Length | Data
   10     123456
    8     0003
   21     ABCDEFGHIJ*****22
    5     *
```

Sort records between a header and trailer

A customer asked us the following question:

> Could you please let me know how to sort a file which has a header and a footer along with detail records. For example:
>
> ```
> HEDRINF101D342005041300000010
> 00000000034  10191610022005030119890717FANDERSON
> 00000000034  21328400022005031019850705FLYONS
> 00000000034  C18176000220050301190000101FMITCHELL
> 00000000034  D19658900520050301200031203MBENTON
> 00000000034  B25291800620050301199940122MGONZALEZ
> 00000000034  42667570102005030119000101MMATHEW
> 00000000034  B28680700520050330199951005MSOLORIO
> 00000000034  C35483500720050301199660930FMACARIO
> 00000000034  F407832003200500322200050318FWILSON
> TRALINF101D3420050413000000010000294138
> ```
>
> I need the data records to be sorted on positions 14-23. If I do a normal sort, the header and footer records end up out of place in the output file. How can I keep the header as the first record and the trailer as the last record, but still sort the data records in between?

In the data shown for this question, the header record is identified by the string 'HEDR' and the trailer record is identified by the string 'TRAL'. We can use those identifiers to keep the header and trailer records in their places as we sort the data records. Here's a DFSORT job that shows how:

```
//S1     EXEC  PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SORTIN DD *
HEDRINF101D342005041300000010
00000000034   101916100220050301198890717FANDERSON
00000000034   213284000220050310198500705FLYONS
00000000034   C18176000220050301190000101FMITCHELL
00000000034   D196589005200503012003120 3MBENTON
00000000034   B25291800620050301199940122MGONZALEZ
00000000034   426675701020050301190000101MMATHEW
00000000034   B286807005200503301995105MSOLORIO
00000000034   C35483500720050301196609930FMACARIO
00000000034   F407832003200503222 00050318FWILSON
TRALINF101D342005041300000001000294138
/*
//SORTOUT   DD DSN=... output file
//SYSIN     DD    *
* Put special key of '1' in 81 for data records.
   INREC IFTHEN=(WHEN=INIT,OVERLAY=(81:C'1')),
* For header record, put special key of '0' in 81.
       IFTHEN=(WHEN=(1,4,CH,EQ,C'HEDR'),OVERLAY=(81:C'0')),
* For trailer record, put special key of '9' in 81.
       IFTHEN=(WHEN=(1,4,CH,EQ,C'TRAL'),OVERLAY=(81:C'9'))
* Sort by special key ('0', '1' or '9') and then regular key.
   SORT FIELDS=(81,1,CH,A,14,10,CH,A)
* Remove special key.
   OUTREC FIELDS=(1,80)
/*
```

By using a special key of '0' for the header record, '1' for the data records and '9' for the trailer records **before** the regular key, we ensure that the header is first and the trailer is last. The '0' special key is unique and will be sorted first so it's regular key doesn't matter. Likewise, the '9' special key for the trailer is unique and will be sorted last so it's regular key doesn't matter. However, since all of the data records have the same special key of '1', they will be sorted between the header and trailer, and the regular key will determine their order.

The output records will look as follows:

```
HEDRINF101D342005041300000010
00000000034   B25291800620050301199940122MGONZALEZ
00000000034   B286807005200503301995105MSOLORIO
00000000034   C18176000220050301190000101FMITCHELL
00000000034   C35483500720050301196609930FMACARIO
00000000034   D196589005200503012003120 3MBENTON
00000000034   F407832003200503222 00050318FWILSON
00000000034   101916100220050301198890717FANDERSON
00000000034   213284000220050310198500705FLYONS
00000000034   426675701020050301190000101MMATHEW
TRALINF101D342005041300000001000294138
```

Keep the last n records

Keeping the first n records of an input data set with DFSORT is pretty easy. You just use the STOPAFT=n operand of OPTION or the ENDREC operand of OUTFIL. For example, to keep the first 10 records, you can use either:

```
  OPTION COPY,STOPAFT=10
```

or:

```
  OPTION COPY
  OUTFIL ENDREC=10
```

But keeping the last n records of an input data set is a bit more challenging since DFSORT doesn't have any built-in functions to do that. The trick is to attach a sequence number to the records and sort them in descending order by the sequence number. That way, the last records end up at the front of the file and you can use STOPAFT or ENDREC to select the last n records. Unfortunately, those last n records will be in reverse order, so you have to sort them again in ascending order by the sequence number to get them back in their original order.

It's less complicated then it sounds as illustrated by this DFSORT/ICETOOL job to get the last 10 records for an input data set with RECFM=FB and LRECL=80:

```
//S1    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//IN DD DSN=...  FB input file
//TEMP DD DSN=&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//OUT DD DSN=...   FB output data set
//TOOLIN DD *
 SORT FROM(IN) USING(CTL1)
 SORT FROM(TEMP) TO(OUT) USING(CTL2)
/*
//CTL1CNTL DD *
* Add sequence numbers. Use them to reverse the order of
* the records.
  INREC FIELDS=(1,80,SEQNUM,8,BI)
  SORT FIELDS=(81,8,BI,D)
* Get the last 10 records
  OUTFIL FNAMES=TEMP,ENDREC=10
/*
//CTL2CNTL DD *
* Use the sequence numbers to put the last 10 records back in their
* original order.
  SORT FIELDS=(81,8,BI,A)
* Remove the sequence numbers.
  OUTREC FIELDS=(1,80)
/*
```

Here's another example. This one gets the last 525 records from a VB input data set:

```
//S2    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//IN DD DSN=...  VB input file
//TEMP DD DSN=&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//OUT DD DSN=...   VB output data set
//TOOLIN DD *
 SORT FROM(IN) USING(CTL1)
 SORT FROM(TEMP) TO(OUT) USING(CTL2)
/*
//CTL1CNTL DD *
* Add sequence numbers. Use them to reverse the order of
* the records.
  INREC FIELDS=(1,4,5:SEQNUM,8,BI,13:5)
  SORT FIELDS=(5,8,BI,D)
* Get the last 525 records
  OUTFIL FNAMES=TEMP,ENDREC=525
/*
//CTL2CNTL DD *
* Use the sequence numbers to put the last 525 records back in their
* original order.
  SORT FIELDS=(5,8,BI,A)
* Remove the sequence numbers.
  OUTREC FIELDS=(1,4,13)
/*
```

Sample records

This question was posed on the MVSHELP Help board:

> I would like to create a subset of a rather large file containing 66 million rows. Not knowing how the data in
> the file is ordered, I'd like to just make a subset of the file by selecting every 100th record. Does anyone know
> of a utility that performs this function?

Below is an ICETOOL job that can sample every 100th record using the SAMPLE=n parameter of OUTFIL.
SAMPLE=n and SAMPLE=(n,m) let you sample records in various ways. In this case STARTREC=100 is used to
write record 100 to SORTOUT, and SAMPLE=100 is used to write records 200, 300, and so on to SORTOUT.

```
//S1 EXEC PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SORTIN DD DSN=...   input file
//SORTOUT DD DSN=...  output file
//SYSIN    DD    *
 OPTION COPY
 OUTFIL STARTREC=100,SAMPLE=100
/*
```

Change all zeros in your records to spaces

The following related questions have been asked by various customers:

- How can I replace all the X'00' bytes in a file with X'40' bytes?

- Is there a way to take a large file and replace the low values (X'00') interspersed throughout the file with
  spaces (X'40') instead? The low values can show up anywhere; they aren't in fixed positions.

- Can you tell me how to replace all the occurrences of one character (say 'a') with another character (say 'b') in a sequential file.

Translation features of INREC, OUTREC and OUTFIL OUTREC make it easy to satisfy all of these requests. The TRAN=ALTSEQ operand can be used to change any character in an input file to another character in the output file within a specified field or throughout the entire record. The ALTSEQ statement is used to specify the from and to characters, and TRAN=ALTSEQ is used to specify which field or fields the ALTSEQ changes are to be applied to.

Here's how you could change all low values (X'00') to spaces (X'40'), in an FB data set with an LRECL of 60:

```
ALTSEQ CODE=(0040)
OUTREC FIELDS=(1,60,TRAN=ALTSEQ)
```

Here's how you could change all 'a' (X'81') and 'x' (X'A7') characters to 'b' (X'82') and 'y' (X'A8') characters, respectively, in a VB input data set with any LRECL:

```
ALTSEQ CODE=(81A7,82A8)
OUTREC FIELDS=(1,4,5,TRAN=ALTSEQ)
```

Of course, you can make your changes to specified fields instead of to the entire record. This comes in handy when you have mixed character and numeric fields in your records and want to avoid making changes to the numeric fields. For example, if you had an FB input file that had characters in bytes 1-20, a PD field in bytes 21-25, and characters in bytes 26-80, you could changes zeros to spaces in the character fields using:

```
ALTSEQ CODE=(0040)
OUTREC FIELDS=(1,20,TRAN=ALTSEQ,   CH - change zeros to spaces
               21,5,               PD field - no change
               26,55,TRAN=ALTSEQ)  CH - change zeros to spaces
```

By not using TRAN=ALTSEQ for the PD field, we avoid changing PD values incorrectly, such as from X'000000001C' (P'1') to X'404040401C' (P'404040401').

Insert date and time of run into records

The following related questions have been asked by various customers:

- Is there any way to dynamically insert the current date, like 'yyyy-mm-dd', in my records? Something like:

```
SORT FIELDS=(1,6,CH,A),FORMAT=CH
OUTREC FIELDS=(1:C'2002-03-11',11:1,6)
```

But with the current date in positions 1-10 instead of a hardcoded constant?

- How can I save the current system date (any format) into a sequential file, in such a way that all the file will ever contain is the said date (this will be used in another step)?

You can use INREC, OUTREC and OUTFIL OUTREC to insert the current date and time in a variety of formats into your records. You can also insert a past date or a future date in a variety of formats into your records.

Both the DATE1(c) and DATE=(4MDc) operands correspond to a C'yyyycmmcdd' constant for today's date where c is any separator character you like except blank. So either of the following pairs of control statements will sort your records on input positions 1-6 and reformat them with today's date in the form C'yyyy-mm-dd' in output positions 1-10, and input positions 1-6 in output positions 11-16.

```
SORT FIELDS=(1,6,CH,A),FORMAT=CH
OUTREC FIELDS=(1:DATE1(-),11:1,6)
```

or

```
    SORT FIELDS=(1,6,CH,A),FORMAT=CH
    OUTREC FIELDS=(1:DATE=(4MD-),11:1,6)
```

You could insert the current time as well as the current date in your records to produce a timestamp. For example:

```
    OUTREC FIELDS=(DATE3,TIME1,1,6)
```

would produce a character timestamp in output positions 1-12 of the form:

yyyydddhhmmss

Date constants can be produced in a variety of other character, zoned decimal and packed decimal formats as well such as C'yyyy-mm', Z'yyyymmdd' and P'yyddd'. Time constants can also be produced in a variety of other character, zoned decimal and packed decimal formats as well such as C'hh:mm', Z'hhmmssxx' and P'hhmmss'.

If, as in the second question above, you wanted to produce just one record containing the date, you could select from a variety of date formats. For example, if you wanted to create a record with just C'dddyy', you could do it with OUTREC as follows:

```
//DATERCD EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD *
DUMMY RECORD
//SORTOUT DD DSN=...
//SYSIN DD *
   OPTION COPY
   OUTREC FIELDS=(YDDDNS=(DY))
/*
```

Change uppercase to lowercase or lowercase to uppercase

A customer asked the following question:

   Can DFSORT be used change a field from lowercase to uppercase?

Translation features of INREC, OUTREC and OUTFIL OUTREC make it easy to change the case of characters in your fields. The TRAN=LTOU operand can be used to change lowercase EBCDIC letters (a-z) to uppercase EBCDIC letters (A-Z) anywhere in a field. The TRAN=UTOL operand can be used to change uppercase EBCDIC letters to lowercase EBCDIC letters anywhere in a field.

Here's how you could change lowercase letters to uppercase letters in a 100-byte character field starting at position 51 and in a 40-byte character field starting in position 301, in an FB data set with an LRECL of 500:

```
  OUTREC FIELDS=(1,50,    1-50:    no change
    51,100,TRAN=LTOU,  51-150:  lowercase to uppercase
    151,150,           151-300: no change
    301,40,TRAN=LTOU,  301-340: lowercase to uppercase
    341,160)           341-500: no change
```

Of course, you could change the case in the entire record as well. For example, here's how you could change uppercase to lowercase in the records of an FB data set with an LRECL of 200:

```
  OUTREC FIELDS=(1,200,TRAN=UTOL)
```

And here's how you could change uppercase to lowercase in the records of a VB data set with any LRECL:

```
  OUTREC FIELDS=(1,4,5,TRAN=UTOL)
```

RACF "SPECIAL" report with and without DFSORT symbols

A customer asked us the following question:

> It is required that ALL activities which are allowed because the user has the SPECIAL attribute be reported and monitored. I am trying to produce a report like this and having some trouble.

> I am looking for something similar to RACF Report Writer with the SELECT PROCESS AUTHORITY(SPECIAL) option but using ICETOOL reports. RACFRW produces the actual commands issued.

> The problem is that some of the fields of different EVENTS reside at different positions. Is there way to solve this?

We worked with the customer and Mark Nelson of the RACF group to produce an ICETOOL job that gave the customer the report he wanted. In fact, at the customer's request, we created two versions of the ICETOOL job; one using the DFSORT symbols for RACF from Mark's RACFICE package, and another without DFSORT symbols. They both perform the same function, but in the symbols version, you can use names for the RACF fields rather than their positions, lengths and formats.

The jobs use several of DFSORT's features, including ICETOOL, IFTHEN clauses and INCLUDE substring logic.

Here are the two versions of the job:

RACF "SPECIAL" report with symbols

```
//RPTS  EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//ADUDATA DD DSN=....    IRRADU00 data
//* RACF Symbols from RACFICE
//SYMNAMES DD DSN=USER01.RACFICE.CNTL(ADUSMBLS),DISP=SHR
//* Temporary symbols
//   DD *
***** Temporary Symbols for this job ******
** MAPPING OF REFORMATTED FIELDS FOR REPORT
RPT_TIME,5,8,CH
RPT_DATE,13,10,CH
RPT_USERID,23,8,CH
RPT_SYSTEM,31,4,CH
RPT_EVENT,35,8,CH
RPT_USERNAME,43,20,CH
RPT_PROF80,63,80,CH
RPT_SPEC80,143,80,CH
** RDW
RDW,1,4
** SHOW FIRST 80 BYTES OF EACH 'LARGE' FIELD
POSITION,AU_SPECIFIED
AU_SPEC80,*,80,CH
POSITION,ALU_SPECIFIED
ALU_SPEC80,*,80,CH
POSITION,DELU_SPECIFIED
DELU_SPEC80,*,80,CH
POSITION,RDEF_SPECIFIED
RDEF_SPEC80,*,80,CH
POSITION,RDEF_RES_NAME
RDEF_RNAME80,*,80,CH
```

```
POSITION,RVAR_SPECIFIED
RVAR_SPEC80,*,80,CH
POSITION,PERM_SPECIFIED
PERM_SPEC80,*,80,CH
POSITION,PERM_RES_NAME
PERM_RNAME80,*,80,CH
POSITION,AD_SPECIFIED
AD_SPEC80,*,80,CH
POSITION,PWD_SPECIFIED
PWD_SPEC80,*,80,CH
POSITION,ACC_RES_NAME
ACC_RNAME80,*,80,CH
/*
//TEMP1 DD DSN=&&T1,DISP=(,PASS),SPACE=(CYL,(2,5),RLSE),UNIT=SYSDA
//REPORT DD SYSOUT=*
//TOOLIN DD *
* Use DFSORT control statements in CTL1CNTL to select only the
* records for users with SPECIAL, reformat different event type
* records to a common format for the report, and sort the
* selected/reformatted records by the User Id field.
 SORT FROM(ADUDATA) TO(TEMP1) USING(CTL1)
* Print the report from the selected/reformatted records.
 DISPLAY FROM(TEMP1) LIST(REPORT) -
   BLANK PAGE -
   TITLE('Commands Executed Due to the SPECIAL Attribute') -
   DATE(4MD/) TIME(12:) -
   ON(RPT_TIME)     HEADER('Time') -
   ON(RPT_DATE)     HEADER('Date') -
   ON(RPT_USERID)   HEADER('User ID') -
   ON(RPT_SYSTEM)   HEADER('System') -
   ON(RPT_USERNAME) HEADER('User Name') -
   ON(RPT_EVENT)    HEADER('Event') -
   ON(RPT_PROF80)   HEADER('Command Target') -
   ON(RPT_SPEC80)   HEADER('Keywords Specified')
/*
//CTL1CNTL DD *
* Select only the records for users with SPECIAL.
   INCLUDE COND=(ACC_AUTH_SPECIAL,EQ,C'YES')
* Reformat different event type records to a common format
* for the report.
  INREC IFTHEN=(WHEN=(PERM_EVENT_TYPE,EQ,C'PERMIT'),
    BUILD=(RDW,5:PERM_TIME_WRITTEN,13:PERM_DATE_WRITTEN,
    23:PERM_EVT_USER_ID,31:PERM_SYSTEM_SMFID,35:PERM_EVENT_TYPE,
    43:PERM_USER_NAME,
    63:PERM_RNAME80,143:PERM_SPEC80)),
   IFTHEN=(WHEN=(AU_EVENT_TYPE,EQ,C'ADDUSER'),
    BUILD=(RDW,5:AU_TIME_WRITTEN,13:AU_DATE_WRITTEN,
    23:AU_EVT_USER_ID,31:AU_SYSTEM_SMFID,35:AU_EVENT_TYPE,
    43:AU_USER_NAME,
    63:AU_USER_ID,143:AU_SPEC80)),
   IFTHEN=(WHEN=(ALU_EVENT_TYPE,EQ,C'ALTUSER'),
    BUILD=(RDW,5:ALU_TIME_WRITTEN,13:ALU_DATE_WRITTEN,
    23:ALU_EVT_USER_ID,31:ALU_SYSTEM_SMFID,35:ALU_EVENT_TYPE,
    43:ALU_USER_NAME,
    63:ALU_USER_ID,143:ALU_SPEC80)),
   IFTHEN=(WHEN=(DELU_EVENT_TYPE,SS,EQ,
    C'DELUSER ,ADDGROUP,ALTGROUP,DELGROUP,CONNECT ,REMOVE  '),
    BUILD=(RDW,5:DELU_TIME_WRITTEN,13:DELU_DATE_WRITTEN,
```

```
       23:DELU_EVT_USER_ID,31:DELU_SYSTEM_SMFID,35:DELU_EVENT_TYPE,
        43:DELU_USER_NAME,
        63:DELU_USER_ID,143:DELU_SPEC80)),
      IFTHEN=(WHEN=(RDEF_EVENT_TYPE,SS,EQ,C'RDEFINE ,RDELETE ,RALTER  '),
       BUILD=(RDW,5:RDEF_TIME_WRITTEN,13:RDEF_DATE_WRITTEN,
        23:RDEF_EVT_USER_ID,31:RDEF_SYSTEM_SMFID,35:RDEF_EVENT_TYPE,
        43:RDEF_USER_NAME,
        63:RDEF_RNAME80,143:RDEF_SPEC80)),
      IFTHEN=(WHEN=(PWD_EVENT_TYPE,EQ,C'PASSWORD'),
       BUILD=(RDW,5:PWD_TIME_WRITTEN,13:PWD_DATE_WRITTEN,
        23:PWD_EVT_USER_ID,31:PWD_SYSTEM_SMFID,35:PWD_EVENT_TYPE,
        43:PWD_USER_NAME,
        63:X,143:PWD_SPEC80)),
      IFTHEN=(WHEN=(RVAR_EVENT_TYPE,SS,EQ,C'RVARY    ,SETROPTS'),
       BUILD=(RDW,5:RVAR_TIME_WRITTEN,13:RVAR_DATE_WRITTEN,
        23:RVAR_EVT_USER_ID,31:RVAR_SYSTEM_SMFID,35:RVAR_EVENT_TYPE,
        43:RVAR_USER_NAME,
        63:X,143:RVAR_SPEC80)),
      IFTHEN=(WHEN=(ACC_EVENT_TYPE,EQ,C'ACCESS'),
       BUILD=(RDW,5:ACC_TIME_WRITTEN,13:ACC_DATE_WRITTEN,
        23:ACC_EVT_USER_ID,31:ACC_SYSTEM_SMFID,35:ACC_EVENT_TYPE,
        43:ACC_USER_NAME,
        63:ACC_RNAME80,143:ACC_EVENT_QUAL)),
      IFTHEN=(WHEN=(AD_EVENT_TYPE,SS,EQ,C'ADDSD    ,ALTDSD  ,DELDSD  '),
       BUILD=(RDW,5:AD_TIME_WRITTEN,13:AD_DATE_WRITTEN,
        23:AD_EVT_USER_ID,31:AD_SYSTEM_SMFID,35:AD_EVENT_TYPE,
        43:AD_USER_NAME,
        63:AD_DS_NAME,143:AD_SPEC80))
* Sort the selected/reformatted records by the User Id field.
   SORT FIELDS=(RPT_USERID,A)
/*
```

RACF "SPECIAL" report without DFSORT symbols

```
//RPTNS   EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//ADUDATA DD DSN=....      IRRADU00 data
//TEMP1 DD DSN=&&T1,DISP=(,PASS),SPACE=(CYL,(2,5),RLSE),UNIT=SYSDA
//REPORT DD SYSOUT=*
//TOOLIN DD *
* Use DFSORT control statements in CTL1CNTL to select only the
* records for users with SPECIAL, reformat different event type
* records to a common format for the report, and sort the
* selected/reformatted records by the User Id field.
 SORT FROM(ADUDATA) TO(TEMP1) USING(CTL1)
* Print the report from the selected/reformatted records.
 DISPLAY FROM(TEMP1) LIST(REPORT) -
   BLANK PAGE -
   TITLE('Commands Executed Due to the SPECIAL Attribute') -
   DATE(4MD/) TIME(12:) -
   ON(5,8,CH)    HEADER('Time') -
   ON(13,10,CH)  HEADER('Date') -
   ON(23,8,CH)   HEADER('User ID') -
   ON(31,4,CH)   HEADER('System') -
   ON(43,20,CH)  HEADER('User Name') -
   ON(35,8,CH)   HEADER('Event') -
   ON(63,80,CH)  HEADER('Command Target') -
```

```
    ON(143,80,CH) HEADER('Keywords Specified')
/*
//CTL1CNTL DD *
* Select only the records for users with SPECIAL.
  INCLUDE COND=(86,3,CH,EQ,C'YES')
* Reformat different event type records to a common format
* for the report.
  INREC IFTHEN=(WHEN=(5,8,CH,EQ,C'PERMIT'),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:304,20,63:507,80,143:763,80)),
   IFTHEN=(WHEN=(5,8,CH,EQ,C'ADDUSER'),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:295,20,63:508,8,143:517,80)),
   IFTHEN=(WHEN=(5,8,CH,EQ,C'ALTUSER'),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:295,20,63:522,8,143:531,80)),
   IFTHEN=(WHEN=(5,8,SS,EQ,
     C'DELUSER ,ADDGROUP,ALTGROUP,DELGROUP,CONNECT ,REMOVE  '),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:295,20,63:498,8,143:507,80)),
   IFTHEN=(WHEN=(5,8,SS,EQ,C'RDEFINE ,RDELETE ,RALTER  '),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:304,20,63:516,80,143:772,80)),
   IFTHEN=(WHEN=(5,8,CH,EQ,C'PASSWORD'),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:295,20,63:X,143:498,80)),
   IFTHEN=(WHEN=(5,8,SS,EQ,C'RVARY   ,SETROPTS'),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:286,20,63:X,143:489,80)),
   IFTHEN=(WHEN=(5,8,CH,EQ,C'ACCESS'),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:1126,20,63:286,80,143:14,8)),
   IFTHEN=(WHEN=(5,8,SS,EQ,C'ADDSD   ,ALTDSD  ,DELDSD  '),
    BUILD=(1,4,5:23,8,13:32,10,23:63,8,31:43,4,35:5,8,
           43:295,20,63:524,44,143:569,80))
* Sort the selected/reformatted records by the User Id field.
  SORT FIELDS=(23,8,CH,A)
/*
```

Multiple output records from some (but not all) input records

A customer asked us the following question:

> I have an FBA input file with an LRECL of 121. It has some records with a '1' (eject) as the ASA carriage control character, followed by data. I want to replace each such record with three records as follows:
>
> - A record with a '1' (eject) in column 1 followed by blanks
>
> - A record with a blank (newline) in column 1 followed by blanks
>
> - A record with a blank (newline) in column 1 followed by the data from the input record
>
> I want to keep records that don't have a '1' in column 1 as is. The output should be FBA with an LRECL of 121.
>
> For example, if my input file contained the following records:

```
1data 01
 data 02
 data 03
1data 04
 data 05
```

I'd want my output file to contain:

```
1

 data 01
 data 02
 data 03
1

 data 04
 data 05
```

Can you help?

You can use an IFTHEN clause in an OUTFIL statement to do this quite easily.

Here's an example of a DFSORT job to create the output file:

```
//MULT EXEC  PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SORTIN DD DSN=...   INPUT FILE
//SORTOUT DD DSN=...  OUTPUT FILE
//SYSIN DD *
* CREATE 3 OUTPUT RECORDS FOR EACH CC '1' INPUT RECORD.
* COPY EACH NON CC '1' RECORD AS IS.
  OPTION COPY
  OUTFIL IFTHEN=(WHEN=(1,1,CH,EQ,C'1'),  CC IS '1'
    BUILD=(C'1',120X,/,    '1',BLANKS
          121X,/,          ' ',BLANKS
          X,2,120))        ' ',DATA
/*
```

Replace leading spaces with zeros

A customer asked us the following question:

I have to replace all leading spaces (X'40') in positions 1 to 6 with zeroes (X'F0'). The input file has
RECFM=FB and LRECL=80 and looks like this:

```
0001XX 20041210
  7 S5
 2044X
  X4407 20041110
X8768978
    25 0003
 XX46464
 646 5665
   FF FDFS
       ABC
     1 QRSTUV
   054   X
 62
```

Non-leading blanks, as for example the fourth character in the 7 S5 record, should **not** be replaced by zeros. So the output file should look like this:

```
0001XX 20041210
007 S5
02044X
00X4407 20041110
X8768978
000025 0003
0XX46464
0646 5665
000FF FDFS
000000  ABC
000001 QRSTUV
000054   X
062
```

Can I use DFSORT or ICETOOL to do this?

Using DFSORT's IFTHEN clauses, we can check for 6-1 leading blanks and replace them with 6-1 leading zeros, respectively, as follows:

```
//S1    EXEC  PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SORTIN DD DSN=...  input file
//SORTOUT DD DSN=...  output file
//SYSIN    DD   *
  OPTION COPY
  INREC IFTHEN=(WHEN=(1,6,CH,EQ,C' '),OVERLAY=(6C'0')),
        IFTHEN=(WHEN=(1,5,CH,EQ,C' '),OVERLAY=(5C'0')),
        IFTHEN=(WHEN=(1,4,CH,EQ,C' '),OVERLAY=(4C'0')),
        IFTHEN=(WHEN=(1,3,CH,EQ,C' '),OVERLAY=(3C'0')),
        IFTHEN=(WHEN=(1,2,CH,EQ,C' '),OVERLAY=(2C'0')),
        IFTHEN=(WHEN=(1,1,CH,EQ,C' '),OVERLAY=(C'0'))
/*
```

Generate JCL to submit to the internal reader

A customer asked the following question:

I have a program that generates a data set with a list of file names that can vary from run to run. I'd like to use ICEGENER to copy the records from the files in the list into one output file. For example, the generated data set might have these file names:

```
RUN001.OUTPUT
RUN001.RESULTS
RUN005.OUTPUT
RUN005.RESULTS
```

so I'd want my output file to have the records from RUN001.OUTPUT, RUN001.RESULTS, RUN005.OUTPUTand RUN005.RESULTS in that order. Can DFSORT help me do this?

We can copy the input data sets with an ICEGENER JOB like the following:

```
//CPYFILES JOB (XXX,005),'PRGMR',CLASS=A,MSGCLASS=H,
// MSGLEVEL=(1,1),TIME=(,15)
//S1 EXEC PGM=ICEGENER
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DSN=&OUT,DISP=(,PASS),SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=file1
//       DD DISP=SHR,DSN=file2
...
//       DD DISP=SHR,DSN=filen
```

But since we have to get the file1, file2, ..., filen names from the data set containing the list of file names, we'll actually have to generate the JCL shown above and have the internal reader submit it for execution. That's the tricky part, but here's what we'll do:

- We'll generate all of the JCL statements, except for the SYSUT1 concatenation statements, with OUTFIL's HEADER1 operand. The HEADER1 output is written before the data records and can contain as many output records as you like (/ is used to start a new record).

- Since HEADER1 is a report parameter, the report data set is normally given a RECFM of FBA and a 1-byte ANSI carriage control character (for example, '1' for page eject) is placed in position 1 of the report. For our purposes, we don't want that ANSI character in position 1 because it would give us invalid JCL (e.g. '1//CPYFILES ...' instead of '//CPYFILES ...". So we'll use OUTFIL's REMOVECC operand remove the ANSI character from each JCL statement we generate.

- We'll generate the SYSUT1 concatenation statements with OUTFIL's OUTREC operand. We'll change the file name in positions 1-44 of each input record into a DD statement referencing that file name. But as shown in the JCL we're trying to generate above, we need 'SYSUT1' as the ddname for the first DD and blanks as the ddname for the second and subsequent DD. To make this happen, we'll use an OUTREC statement with a SEQNUM operand to add a sequence number to the input records, and then use CHANGE in OUTFIL OUTREC to determine if the sequence number is 1 or not so we can set up the correct ddname.

Here's our DFSORT job to generate the ICEGENER JCL we need and have the internal reader submit it:

```
//GENJOB EXEC  PGM=ICEMAN
//SYSOUT DD   SYSOUT=*
//SORTIN DD DSN=...          list of file names
//IRDR DD SYSOUT=(A,INTRDR)  internal reader
//SYSIN DD *
  OPTION COPY
* Add sequence numbers to file list so we can identify the
* first file in the list and use '//SYSUT1 DD' for it.
* We'll use '// DD' for the second and subsequent files
* in the list.
  OUTREC FIELDS=(1,44,     file name from list
     51:SEQNUM,3,ZD)        sequence number
* Generate the JCL for output to the internal reader
  OUTFIL FNAMES=IRDR,
* Remove the carriage control character from the "report".
    REMOVECC,
* Generate JOB, EXEC, SYSPRINT, SYSUT2 and SYSIN statements.
    HEADER1=('//CPYFILES JOB (XXX,005),''PRGMR'',',
     'CLASS=A,MSGCLASS=H,',/,
     '// MSGLEVEL=(1,1),TIME=(,15)',/,
     '//S1 EXEC PGM=ICEGENER',/,
     '//SYSPRINT DD SYSOUT=*',/,
     '//SYSUT2 DD DSN=&OUT,DISP=(,PASS),SPACE=(CYL,(5,5)),',
     'UNIT=SYSDA',/,
     '//SYSIN DD DUMMY'),
* Generate //SYSUT1 DD DISP=SHR,DSN=name1
*          //       DD DISP=SHR,DSN=name2
*          ...
    OUTREC=(C'//',
     51,3,CHANGE=(6,C'001',C'SYSUT1'),  'SYSUT1' for first name
          NOMATCH=(C' '),               blanks for subsequent names
     C' DD DISP=SHR,DSN=',
     1,44,                              file name from list
     80:X)                              ensure record length is 80

/*
```

If you'd rather not encode your JOB, etc statements as constants, you can do the same thing in several DFSORT
steps that we did above in one DFSORT step, like this:

```
//GENJCL1 EXEC  PGM=ICEMAN
//SYSOUT DD   SYSOUT=*
//SORTIN DD DATA,DLM=$$
//CPYFILES JOB (XXX,005),'PRGMR',CLASS=A,MSGCLASS=H,
// MSGLEVEL=(1,1),TIME=(,15)
//S1 EXEC PGM=ICEGENER
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DSN=&OUT,DISP=(,PASS),SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN DD DUMMY
$$
//SORTOUT DD DSN=&T1,UNIT=SYSDA,SPACE=(CYL,(1,1)),DISP=(,PASS)
//SYSIN    DD *
* Copy JOB, EXEC, SYSPRINT, SYSUT2 and SYSIN statements.
  OPTION COPY
/*
//GENJCL2 EXEC  PGM=ICEMAN
//SYSOUT   DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
```

```
//SORTIN DD DSN=...          list of file names
//TEMP DD DSN=&T1,DISP=(MOD,PASS)
//SYSIN   DD    *
 OPTION COPY
* Add sequence numbers to file list so we can identify the
* first file in the list and use '//SYSUT1 DD' for it.
* We'll use '// DD' for the second and subsequent files
* in the list.
  OUTREC FIELDS=(1,44,    file name from list
     51:SEQNUM,3,ZD)       sequence number
* Generate //SYSUT1 DD DISP=SHR,DSN=name1
*          //       DD DISP=SHR,DSN=name2
*          ...
  OUTFIL FNAMES=TEMP,
    OUTREC=(C'//',
    51,3,CHANGE=(6,C'001',C'SYSUT1'),  'SYSUT1' for first name
        NOMATCH=(C' '),               blanks for subsequent names
    C' DD DISP=SHR,DSN=',
    1,44,                             file name from list
    80:X)                             ensure record length is 80
/*
//SUBJCL EXEC PGM=ICEMAN
//SYSOUT DD  SYSOUT=*
//SORTIN DD DSN=&T1,DISP=(OLD,PASS)
//SORTOUT DD SYSOUT=(A,INTRDR)        internal reader
//SYSIN   DD *
* Submit the JCL to the internal reader
  OPTION COPY
/*
```

Totals by key and grand totals

A customer asked the following question:

I have a file with ZD data for which I want totals by key and grand totals. The 10-byte CH key starts in position 1 and the 12-byte ZD field starts in position 19. Here's an example of what the input might look like:

```
(Key)           (Data)
052500ABCDGRC    00000002246K
053500ABCDGRC    00000828784K
054500ABCDGRC    00000000155H
052501ABCDGRC    02019572110B
053501ABCDGRC    00121859976B
054501ABCDGRC    00515208642F
052502ABCDGRC    02638364444R
053502ABCDGRC    00131222671P
054502ABCDGRC    00824793989R
```

And here's what I'd want the sorted output to look like:

```
(Key)           (052 total)   (053 total)   (054 total)   (grand total)
500ABCDGRC      00000002246K  00000828784K  00000000155H  000008308740
501ABCDGRC      02019572110B  00121859976B  00515208642F  02656640729{
502ABCDGRC      02638364444R  00131222671P  00824793989R  03594381106N
```

Note that the (headings) shown above are just for reference; they shouldn't actually appear in the output data set.

I know an indirect method of doing this which will take 5 passes over the data, but is there a way to do it in a single pass with DFSORT?

The trick here is to use IFTHEN clauses to reformat the records for each type of key (052, 053 and 054) with slots for all of the keys and the grand total. The actual key will go into its slot and the grand total slot, and zeros will go into the slots for the other keys. That will allow us to use SUM to produce the key totals and grand totals. Here's a DFSORT job that produces the requested output:

```
//S1 EXEC PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SORTIN DD DSN=...  input file
//SORTOUT DD DSN=...  output file
//SYSIN DD *
* Reformat 052 records with a slot for 052, 053, 054 and grand total.
* 052 and total will have the 052 values.  Others will be Z'0'.
  INREC IFTHEN=(WHEN=(1,3,CH,EQ,C'052'),
    BUILD=(1:4,10,16:19,12,30:12C'0',44:12C'0',58:19,12)),
* Reformat 053 records with a slot for 052, 053, 054 and grand total.
* 053 and total will have the 053 values.  Others will be Z'0'.
    IFTHEN=(WHEN=(1,3,CH,EQ,C'053'),
    BUILD=(1:4,10,16:12C'0',30:19,12,44:12C'0',58:19,12)),
* Reformat 054 records with a slot for 052, 053, 054 and grand total.
* 054 and total will have the 054 values.  Others will be Z'0'.
    IFTHEN=(WHEN=(1,3,CH,EQ,C'054'),
    BUILD=(1:4,10,16:12C'0',30:12C'0',44:19,12,58:19,12))
* Sort on the key
  SORT FIELDS=(1,10,CH,A)
* Sum the 052, 053, 054 and grand total fields.
  SUM FORMAT=ZD,FIELDS=(16,12,30,12,44,12,58,12)
/*
```

Omit data set names with Axxx. as the high level qualifier

Radoslaw Skorupka kindly contributed this trick. Here's what it does:

This DFSORT job omits certain data set names from a list. The data sets to be omitted are those that start with 'Axxx.' (where x is any character), except data sets that start with 'ASMA.' or 'ASMT.' are to be kept.

For example, if the input file contained the following records:

```
ASMT.DASD
BILL.MASTER
ALL.DASD
A123.MASTER.IN
ALLOC.DASD
ASMQ.ALL.FILES
SYS1.LINKLIB
A1.MYFILE
ASMA.MYFILE
ALEX.FILE1.OUT
```

We'd want the output file to contain:

```
ASMT.DASD
BILL.MASTER
ALL.DASD
ALLOC.DASD
SYS1.LINKLIB
A1.MYFILE
ASMA.MYFILE
```

Here's Radoslaw's job:

```
//S1  EXEC  PGM=ICEMAN
//SYSOUT    DD  SYSOUT=*
//SORTIN DD DSN=...   INPUT FILE
//SORTOUT DD DSN=...  OUTPUT FILE
//SYSIN DD *
 OPTION COPY
 OMIT FORMAT=CH,                All fields have CH format
   COND=((1,5,NE,C'ASMA.',AND,  Keep if 'ASMA.'
         1,5,NE,C'ASMT.'),AND,  Keep if 'ASMT.'
        (1,1,EQ,C'A',AND,       Keep if not 'A'
         2,1,NE,C'.',AND,       Keep if 'A.'
         3,1,NE,C'.',AND,       Keep if 'Ax.'
         4,1,NE,C'.',AND,       Keep if 'Axx.'
         5,1,EQ,C'.'))          Omit if 'Axxx.'
/*
```

Dataset counts and space by high level qualifier

A customer asked us the following question:

> I have a DCOLLECT file from which I'm stripping the D records for certain high level qualifiers. I'm trying to produce a report showing the number of data sets and the amount of space allocated for each high level qualifier. The space allocation field is (93,4,BI) but I'm not sure how to total this field and give a report such as:
>
> ```
> HLQ          # DATASETS    SPACE USED (Mb)
> ---------    ----------    ---------------
> PAK#              3                   2510
> PAL#              4                   5005
> PAN#              2                    238
> ```
>
> Can you help?

This kind of summarized report requires the use of the SECTIONS and NODETAIL operands of DFSORT's OUTFIL statement. SECTIONS allows us to get counts and totals for each key value, and NODETAIL allows us to show only header or trailer records without the detailed data records.

Here's an example of a DFSORT job to produce this kind of report:

```
//DRPT  EXEC  PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SORTIN DD DSN=...    DCOLLECT file
//DSNOUT  DD SYSOUT=*  Report
//SYSIN   DD    *
* Sort by the HLQ
 SORT FIELDS=(29,4,CH,A)
* Include only D-type records
  INCLUDE COND=(9,1,CH,EQ,C'D',AND,
    29,4,SS,EQ,C'PAK#,PAL#,PAN#')
* Create report
 OUTFIL FNAMES=DSNOUT,
   NODETAIL,
   HEADER2=(1:PAGE,12:DATE,24:'BREAKDOWN BY HLQ',/,
           1:10X,/,
           1:'HLQ',14:'# DATASETS',28:'SPACE USED (Mb)',/,
           1:'---------',14:'----------',28:'---------------'),
   SECTIONS=(29,4,
     TRAILER3=(1:29,4,16:COUNT,33:TOT=(93,4,BI,M10)))
```

Delete duplicate SMF records

A customer asked the following question on the IBM-MAIN mailing list:

Can anyone show me a quick and dirty way to delete duplicate SMF records? Maybe with ICETOOL ??

Here's our response detailing a DFSORT trick that worked for this customer:

You could pad the records (with VLFILL=byte) up to the length you need to identify the duplicates, and then remove the padding (with VLTRIM=byte for output, using the ICETOOL job below. We're using 256 bytes and X'AA' padding for the example, but you can go up to 4084 bytes and use whatever padding character you like that's not likely to be found in the actual SMF data.

```
//DELDUP   EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//IN DD DSN=...  SMF input file
//TEMP DD DSN=&&X1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//OUT DD DSN=...  output file
//TOOLIN    DD    *
* PAD RECORDS LT 256 BYTES UP TO 256 BYTES.
  COPY FROM(IN) USING(CTL1)
* SORT RECORDS, ELIMINATE DUPLICATES, ELIMINATE PADDING.
  SORT FROM(TEMP) USING(CTL2)
/*
//CTL1CNTL DD *
  OUTFIL FNAMES=TEMP,OUTREC=(1,256),VLFILL=X'AA'
/*
//CTL2CNTL DD *
  SORT FIELDS=(1,256,BI,A)
  SUM FIELDS=NONE
  OUTFIL FNAMES=OUT,VLTRIM=X'AA'
```

Sort ddmonyy dates

Alan C. Field asked the following question on the IBM-MAIN mailing list:

> I have a dataset with dates in the form 03Jan97, 23May99, 18Jan00. I would like to sort these records descending by date:
>
> ```
> SORT FIELDS=(6,2,Y2C,D,?????,1,2,CH,D)
> ```
>
> but I need a clever way to get the months in order.
>
> Right now I use a report writer to change JAN to 001, FEB to 002 etc, SORT and then translate back.

DFSORT doesn't have a built-in function for interpreting month names as numeric values, but you can do your SORT of ddmonyy dates directly with DFSORT instead of surrounding it with report writer steps. Just use INREC and OUTREC statements to translate the month back and forth. Here's an example assuming that the ddmonyy date is in positions 1-7 and the input data set has RECFM=FB and LRECL=80.

```
INREC FIELDS=(3,3,        Save MON
   3,3,                   Convert MON to MM
    CHANGE=(2,C'Jan',C'01',C'Feb',C'02',C'Mar',C'03',C'Apr',C'04',
             C'May',C'05',C'Jun',C'06',C'Jul',C'07',C'Aug',C'08',
             C'Sep',C'09',C'Oct',C'10',C'Nov',C'11',C'Dec',C'12'),
   1,2,                   DD
   6,2,                   YY
   8,73)                  73 is length to end of record
 SORT FIELDS=(4,6,Y2W,D) Sort MMDDYY using Century Window
 OUTREC FIELDS=(6,2,      DD
                1,3,      MON
                8,2,      YY
                10,73)    Rest of record
```

Turn cache on for all volumes

Ken Leidner kindly contributed this trick. Here's how he describes it:

This is a DFSORT job that runs a DCOLLECT to get all volumes in the shop and generate the IDCAMS control statements to turn cache on for each one (DEVICE and DASDFASTWRITE). My only problem was getting the SUBSYSTEM and NVS statements generated correctly. But, I like the solution. It also shows how DFSORT can do a lot of things besides just sorting.

We run this at each IPL - just to be sure.

Here's Ken's job:

```
//SETCACHE JOB ...
//STEP1  EXEC  PGM=IEBUPDTE,PARM=NEW,REGION=6M
//*
//*  SET UP DCOLLECT AND DFSORT CONTROL STATEMENTS
//*
//SYSPRINT DD  SYSOUT=*
//SYSUT2   DD  DSN=&&PDS,DISP=(NEW,PASS),
//             LRECL=80,BLKSIZE=0,RECFM=FB,
//             SPACE=(TRK,(2,2,1))
//SYSIN    DD  *
./  ADD  NAME=DCOLLECT
```

```
       DCOLLECT VOLUMES(*) NODATAINFO OUTFILE(OUT)
./  ADD  NAME=SORT
  INCLUDE COND=(9,1,CH,EQ,C'V')
  SORT FIELDS=(81,2,BI,A)
    OUTFIL FNAMES=SORTOUT,CONVERT,OUTREC=(1:
            1C'  SETCACHE VOLUME(',
            29,6,1C') UNIT(3390) DASDFASTWRITE ON  /* ',
            81,2,66:15X)
    OUTFIL FNAMES=SORTOUT2,CONVERT,OUTREC=(1:
            1C'  SETCACHE VOLUME(',
            29,6,1C') UNIT(3390) DEVICE       ON  /* ',
            81,2,66:15X)
    OUTFIL FNAMES=SORTOUT3,INCLUDE=(82,1,BI,NONE,X'0F'),
            CONVERT,OUTREC=(1:
            1C'  SETCACHE VOLUME(',
            29,6,1C') UNIT(3390) SUBSYSTEM    ON  /* ',
            81,2,66:15X)
    OUTFIL FNAMES=SORTOUT4,INCLUDE=(82,1,BI,NONE,X'0F'),
            CONVERT,OUTREC=(1:
            1C'  SETCACHE VOLUME(',
            29,6,1C') UNIT(3390) NVS          ON  /* ',
            81,2,66:15X)
./  ADD  NAME=MERGE
    MERGE FIELDS=(59,4,BI,A,38,6,CH,D)
    OUTFIL FNAMES=SORTOUT,OUTREC=(1,58,59,2,HEX,
            1C' */',66:15X)
/*
//STEP2 EXEC  PGM=IDCAMS,REGION=6M
//*
//*  RUN DCOLLECT ONLY VOLUME INFORMATION
//*
//OUT      DD  DSN=&&DOUT,DISP=(NEW,PASS),
//             SPACE=(TRK,(15,5),RLSE),
//             DSORG=PS,RECFM=VB,LRECL=644,BLKSIZE=0
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  DSN=&&PDS(DCOLLECT),DISP=(OLD,PASS)
//STEP3 EXEC PGM=SORT,REGION=6M
//*
//*  ONLY V RECORDS - OUTPUT VOLSER AND UNIT ADDRESS HEX
//*
//SYSOUT   DD  SYSOUT=*
//SORTIN   DD  DSN=&&DOUT,DISP=(OLD,PASS)
//SORTOUT  DD  DSN=&&SORT1,DISP=(NEW,PASS),
//             SPACE=(TRK,15),LRECL=80,BLKSIZE=0,RECFM=FB
//SORTOUT2 DD  DSN=&&SORT2,DISP=(NEW,PASS),
//             SPACE=(TRK,15),LRECL=80,BLKSIZE=0,RECFM=FB
//SORTOUT3 DD  DSN=&&SORT3,DISP=(NEW,PASS),
//             SPACE=(TRK,15),LRECL=80,BLKSIZE=0,RECFM=FB
//SORTOUT4 DD  DSN=&&SORT4,DISP=(NEW,PASS),
//             SPACE=(TRK,15),LRECL=80,BLKSIZE=0,RECFM=FB
//SYSIN    DD  DSN=&&PDS(SORT),DISP=(OLD,PASS)
//STEP4 EXEC PGM=SORT,REGION=6M
//*
//*  MERGE IDCAMS CONTROL STATEMENTS
//*
//SYSOUT   DD  SYSOUT=*
//SORTIN3  DD  DSN=&&SORT1,DISP=(OLD,DELETE)
//SORTIN4  DD  DSN=&&SORT2,DISP=(OLD,DELETE)
```

```
//SORTIN1  DD  DSN=&&SORT3,DISP=(OLD,DELETE)
//SORTIN2  DD  DSN=&&SORT4,DISP=(OLD,DELETE)
//SORTOUT  DD  DSN=&&SET,DISP=(NEW,PASS),
//             SPACE=(TRK,15),LRECL=80,BLKSIZE=0,RECFM=FB
//SYSIN    DD  DSN=&&PDS(MERGE),DISP=(OLD,PASS)
//*
//*  EXECUTE THE GENERATED IDCAMS CONTROL STATEMENTS
//*
//STEP5  EXEC PGM=IDCAMS,REGION=6M
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  DSN=&&SET,DISP=(OLD,DELETE)
```

Here's a sample of what the generated IDCAMS control statements might look like:

.
.
.
```
  SETCACHE VOLUME(PPPACK) UNIT(3390) DEVICE        ON  /* 0803 */
  SETCACHE VOLUME(PPPACK) UNIT(3390) DASDFASTWRITE ON  /* 0803 */
  SETCACHE VOLUME(USRPAK) UNIT(3390) DEVICE        ON  /* 0808 */
  SETCACHE VOLUME(USRPAK) UNIT(3390) DASDFASTWRITE ON  /* 0808 */
  SETCACHE VOLUME(1P0301) UNIT(3390) DEVICE        ON  /* 0813 */
  SETCACHE VOLUME(1P0301) UNIT(3390) DASDFASTWRITE ON  /* 0813 */
```
.
.
.

C/C++ calls to DFSORT and ICETOOL

DFSORT and ICETOOL can add lots of tricks to your C/C++ programs. Both DFSORT and ICETOOL can be
called from C and C++. The key is to use the system() function in the C/C++ program and supply the needed JCL
and control statements when you execute the program.

Here's a simple example of a C program that calls DFSORT:

```c
/* This example illustrates how to use SYSTEM() to call DFSORT.  */

#include <stdlib.h>
#include <stdio.h>

main()
{
    int dfsrtrc;
    dfsrtrc = system("PGM=SORT");

    if (dfsrtrc >= 16)
      printf("DFSORT failed - RC=%d\n", dfsrtrc);
    else
      printf("DFSORT completed successfully - RC=%d\n", dfsrtrc);

    return(dfsrtrc);
}
```

and here's the JCL you might use to execute this program:

```
//SORTRUN  EXEC PGM=DFSC
//STEPLIB  DD  DSN=...
//STDIN    DD  DUMMY
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SORTIN   DD DSN=...
//SORTOUT  DD DSN=...
//SYSIN    DD *
  SORT FIELDS=(5,2,CH,A)
  OMIT COND=(10,4,CH,EQ,C'DATA')
/*
```

Here's a simple example of a C program that calls ICETOOL:

```
/* This example illustrates how to use SYSTEM() to call ICETOOL.  */

#include <stdlib.h>
#include <stdio.h>

main()
{
    int toolrc;
    toolrc = system("PGM=ICETOOL");

    if (toolrc >= 12)
      printf("ICETOOL failed - RC=%d", toolrc);
    else
      printf("ICETOOL completed successfully - RC=%d", toolrc);

    return(toolrc);
}
```

and here's the JCL you might use to execute this program:

```
//TOOLRUN  EXEC PGM=TOOLC
//STEPLIB  DD  DSN=...
//STDIN    DD  DUMMY
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//TOOLMSG  DD  SYSOUT=*
//DFSMSG   DD  SYSOUT=*
//IN   DD  DSN=...
//OUT1 DD  DSN=...
//OUT2 DD  DSN=...
//TOOLIN   DD *
   SELECT FROM(IN) TO(OUT1) ON(5,2,CH) LAST
   SELECT FROM(IN) TO(OUT2) ON(21,5,ZD) HIGHER(4)
/*
```

REXX calls to DFSORT and ICETOOL

DFSORT and ICETOOL can add lots of tricks to your REXX programs. Both DFSORT and ICETOOL can be called from REXX. The key is to specify allocate statements for the data sets you need and then use a call statement like this:

```
ADDRESS LINKMVS ICEMAN
```

You can set up the required control statements ahead of time or have your REXX program create them.

Here's an example of a REXX CLIST to call DFSORT:

```
/* SIMPLE REXX CLIST TO EXECUTE DFSORT */
/* ASSUMES DFSORT IS IN SYS1.LPALIB */
  "FREE FI(SYSOUT SORTIN SORTOUT SYSIN)"
  "ALLOC FI(SYSOUT)   DA(*)"
  "ALLOC FI(SORTIN)   DA('userid.INS1') REUSE"
  "ALLOC FI(SORTOUT)  DA('userid.OUTS1') REUSE"
  "ALLOC FI(SYSIN)    DA('userid.SORT.STMTS') SHR REUSE"
  ADDRESS LINKMVS ICEMAN
```

and here's the control statement that might appear in userid.SORT.STMTS:

```
  SORT FIELDS=(5,4,CH,A)
```

Here's an example of a REXX CLIST to call ICETOOL:

```
/* SIMPLE REXX CLIST TO EXECUTE ICETOOL */
/* ASSUMES DFSORT IS IN SYS1.LPALIB */
  "FREE FI(TOOLMSG DFSMSG VLR LENDIST TOOLIN)"
  "ALLOC FI(TOOLMSG)   DA(*)"
  "ALLOC FI(DFSMSG)    DUMMY"
  "ALLOC FI(VLR)     DA('userid.VARIN') REUSE"
  "ALLOC FI(LENDIST)   DA(*)"
  "ALLOC FI(TOOLIN)    DA('userid.TOOLIN.STMTS') SHR REUSE"
  ADDRESS LINKMVS ICETOOL
```

and here's the control statements that might appear in userid.TOOLIN.STMTS:

```
 STATS FROM(VLR) ON(VLEN)
 OCCURS FROM(VLR) LIST(LENDIST) -
   TITLE('LENGTH DISTRIBUTION REPORT') BLANK -
   HEADER('LENGTH')  HEADER('NUMBER OF RECORDS') -
   ON(VLEN)          ON(VALCNT)
```

Pull records from a Master file

A customer asked us the following question:

> We are trying to pull a large number of records (2 million accounts) from a large data set on the mainframe. I have one file that contains these 2 million account numbers. Can I do this using DFSORT? It's like a table lookup function, except in this case, the table is just a bit on the huge side. Thanks for any insight you can provide me.

Wow, two million account records - the usual method of using INCLUDE conditions to pull records isn't going to work here!

Questioning the customer, we established that each account number only appeared once in the Master file and once in the Pull file. Given that, we were able to come up with a smart DFSORT trick using the FIRSTDUP operand of ICETOOL's SELECT operator.

Here's the ICETOOL job that does the trick. The Pull file contains the account numbers to "pull" from the Master file. For simplicity, we're assuming both files are fixed-length, although the trick would work for variable-length files with some minor adjustments.

```
//PULLRCDS JOB ...
//SUPULL   EXEC  PGM=ICETOOL
//TOOLMSG   DD   SYSOUT=*
//DFSMSG    DD   SYSOUT=*
//PULL DD DSN=...  the Pull file
//TPULL DD DSN=&T1,UNIT=SYSDA,DISP=(,PASS),SPACE=(CYL,(5,5))
//CONCAT  DD DSN=...  Master file
//        DD DSN=*.TPULL,VOL=REF=*.TPULL,DISP=(OLD,PASS)
//OUTPUT DD DSN=...  Output file with "pulled" records
//TOOLIN DD *
* Reformat the Pull file so the account number is in the same
* place as in the Master file
 COPY FROM(PULL) TO(TPULL) USING(CPY1)
** SELECT the records in the Master File that are also in the PULL file
**   x and n are as above
 SELECT FROM(CONCAT) TO(OUTPUT) ON(x,n,CH) FIRSTDUP
//CPY1CNTL DD *
**
** In the OUTREC statement below:
**   x = position of account number in Master file
**   y = position of account number in Pull file
**   n = length of account number
**   m = LRECL of Master File
**
 OUTREC FIELDS=(x:y,n,m:X)
/*
```

There are a couple of tricks here that merit explanation:

- In order to use SELECT with FIRSTDUP, the keys for both files must be in the same place. We use a COPY operator with an OUTREC statement to create a temporary copy of the Pull file that has the account number in the same place as in the Master file.

- We concatenate the Master file and the temporary Pull file together, in that order, and use SELECT with FIRSTDUP to keep only the records from the Master file with account numbers that appear in both the Master file and the temporary Pull file.

Tricky, but effective :-)

Concurrent VSAM/non-VSAM load

Dave Betten actually contributed this Smart DFSORT Trick before he joined the DFSORT Team (thanks, Dave). It uses OUTFIL to load a VSAM alternate index (with DFSORT instead of REPRO) and a tape data set, concurrently. Dave says:

We were doing a study to tune a customer's batch jobs and came across two separate "problem" jobs where DFSORT's OUTFIL was a great solution!

Job 1 originally consisted of the following steps:

* Step 1: 3 sec - An alternate index is deleted and defined.

* Step 2: 35 min - An application program is used to extract records from a VSAM cluster and write them to a tape data set which is sent offsite.

* Step 3: 41 min - DFSORT is used to sort the extracted data set to a DASD data set.

* Step 4: 16 min - REPRO is used to load the sorted DASD data set into the alternate index.

Our phase 1 solution was to eliminate the REPRO step. We changed step 3 to have DFSORT sort the extracted data set directly into the alternative index. Eliminating step 4 saved about 16 minutes.

Our phase 2 solution was to replace the application program in step 2 with a DFSORT step which extracted the needed records (using INCLUDE), sorted them and used OUTFIL to write the records concurrently to the alternate index and tape data set. By redesigning the job, we:

* Reduced the elapsed time, CPU time and EXCPs significantly.

* Eliminated the application program and its associated maintenance.

* Eliminated the DASD intermediate data set.

* Reduced the number of tape mounts.

Job 2 consisted of the following steps:

* Step 1: 3 sec - A VSAM cluster is deleted and defined.

* Step 2: 16 min - DFSORT is used to sort the output of a batch process to a DASD data set.

* Step 3: 13 min - DFSORT is used to copy the sorted DASD data set to a tape data set which is sent offsite.

* Step 4: 10 min - REPRO is used to load the sorted DASD data set into the VSAM cluster.

Our solution was to use DFSORT's OUTFIL to write the tape data set and load the VSAM cluster concurrently. By redesigning the job, we reduced the elapsed time, CPU time and EXCPs significantly and eliminated the DASD intermediate data set.

Here's a job similar to the revised Job 2 described above:

```
//LDVKSDS  JOB ...
//CREATEIT  EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 DELETE userid.VKSDS PURGE CLUSTER
    IF MAXCC = 8 THEN SET MAXCC = 0
 DEFINE                          /* DEFINE THE MAIN DB     */ -
   CLUSTER(NAME(userid.VKSDS) -
     DATACLASS(COMV)             /* SET DATACLASS          */ -
     CISZ(24576)                 /* MIN VALUE IS 1024      */ -
     CYLINDERS(500 200)          /* LIMITS THE DB SIZE     */ -
     RECSZ(150 1500) FSPC(1 1)   /* VAR. KSDS, FREE SPACE  */ -
     KEYS(100 0)                 /* KEY                    */ -
     SHR(2) IXD SPEED NOREUSE NREPL /* PARAMETERS          */ -
     BUFSPC(1000000))            /* BUFFER SPACE           */ -
   DATA(NAME(userid.VKSDS.DATA)) -
   INDEX(NAME(userid.VKSDS.INDEX) CISZ(4096))
/*
//*
//LOADIT   EXEC PGM=ICEMAN,REGION=8192K
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SORTIN   DD   DSN=userid.VINPUT,DISP=SHR
//KSDSOUT  DD   DSN=userid.VKSDS,DISP=OLD
//TAPEOUT  DD   DSN=userid.VTAPE,...
//SYSIN    DD   *
   SORT FIELDS=(5,100,BI,A)
   OUTFIL FNAMES=(KSDSOUT,TAPEOUT)
/*
```

Important note:

The DEFINE CLUSTER has KEYS(100 0), but the SORT statement has FIELDS=(5,100,...). 100 is the length in both cases, but why does the key start at 0 for the DEFINE CLUSTER and at 5 for the SORT statement? Two reasons:

1. DEFINE CLUSTER uses an **offset** for the key whereas DFSORT uses a **position** for the key. Position 1 is equivalent to offset 0.

2. The SORTIN data set is **variable-length** so its records start with a 4-byte RDW. Thus, from DFSORT's point of view, the key starts at **position 5** after the RDW. DFSORT removes the RDW when it writes the record to the VSAM data set.

    If the SORTIN data set was **fixed-length,** its records would not have RDWs and the key would start at **position 1** from DFSORT's point of view.

DCOLLECT conversion reports

ICETOOL and OUTFIL add lots of tricks to your DFSORT toolkit, both individually and in combination, that you can use to analyze data created by DFHSM, DFSMSrmm, DCOLLECT, SMF, etc.

To illustrate, here's an example of an ICETOOL job that uses DCOLLECT output to produce several conversion reports on the migration status of various volumes. These reports can be helpful when you are converting non-SMS-managed volumes to SMS-managed volumes and need to determine which are still "in conversion" so you can fix them.

To make the reports easy to read, OUTFIL's lookup and change feature is used to assign English descriptions to DCOLLECT's bit flags.

The Storage Administrator Examples (ICESTGEX) available on the DFSORT product tape contains this example as well as many others.

```
//DCOLEX2  JOB ...
//**************************************************************
//* DCOLLECT EXAMPLE 2: CONVERSION REPORTS
//*   ASSIGN ENGLISH DESCRIPTIONS TO BIT FLAGS TO SHOW MIGRATION
//*   STATUS OF VOLUMES (IN CONVERSION, MANAGED BY SMS AND
//*   NON-SMS MANAGED).
//**************************************************************
//STEP1  EXEC  PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*  ICETOOL MESSAGES
//DFSMSG   DD SYSOUT=*  DFSORT  MESSAGES
//TOOLIN   DD *         CONTROL STATEMENTS
* PART 1 - ADD IDENTIFYING STATUS (FLAG) DESCRIPTION FOR
*          'MANAGED', 'IN CONVERSION' AND 'NON-MANAGED'
*          VOLUME RECORDS
  COPY FROM(DCOLALL) USING(FLAG)
* PART 2 - PRINT REPORT SHOWING COUNT OF EACH STATUS TYPE
  OCCUR FROM(STATUS) LIST(REPORTS) -
   TITLE('STATUS COUNT REPORT') DATE -
   BLANK -
   HEADER('STATUS') ON(7,20,CH) -
   HEADER('NUMBER OF VOLUMES') ON(VALCNT)
* PART 3 - PRINT REPORT SORTED BY VOLUMES AND STATUS
  SORT FROM(STATUS) TO(SRTVOUT) USING(SRTV)
  DISPLAY FROM(SRTVOUT) LIST(REPORTV) -
   TITLE('VOLUME/STATUS REPORT') DATE PAGE -
   BLANK -
   HEADER('VOLUME') ON(1,6,CH) -
   HEADER('STATUS') ON(7,20,CH)
* PART 4 - PRINT REPORT SORTED BY STATUS AND VOLUMES
  SORT FROM(STATUS) TO(SRTIOUT) USING(SRTI)
  DISPLAY FROM(SRTIOUT) LIST(REPORTI) -
   TITLE('STATUS/VOLUME REPORT') DATE PAGE -
   BLANK -
   HEADER('STATUS') ON(7,20,CH) -
   HEADER('VOLUME') ON(1,6,CH)
//DCOLALL DD DSN=Y176398.R12.DCOLLECT,DISP=SHR
//STATUS   DD DSN=&&TEMP1,DISP=(,PASS),UNIT=SYSDA,
// LRECL=50,RECFM=FB,DSORG=PS
//SRTVOUT  DD DSN=&&TEMP2,DISP=(,PASS),UNIT=SYSDA
//SRTIOUT  DD DSN=&&TEMP3,DISP=(,PASS),UNIT=SYSDA
//FLAGCNTL DD *
* FIND V-TYPE RECORDS WITH STATUS FLAGS OF INTEREST
  INCLUDE COND=(9,2,CH,EQ,C'V ',AND,
               35,1,BI,NE,B'......10')
* CREATE RECFM=FB OUTPUT RECORDS WITH VOLSER AND
* STATUS DESCRIPTION.
* LOOKUP/CHANGE TABLE -
*   FLAG          DESCRIPTION
*   ----------    --------------------
*   DCVMANGD      MANAGED BY SMS
*   DCVINITL      IN CONVERSION TO SMS
*   DCVNMNGD      NON-SMS MANAGED
```

```
   OUTFIL FNAMES=STATUS,CONVERT,
     OUTREC=(29,6,
       35,1,CHANGE=(20,
             B'......11',C'MANAGED BY SMS',
             B'......01',C'IN CONVERSION TO SMS',
             B'......00',C'NON-SMS MANAGED'),
       50:X)
//REPORTS DD SYSOUT=*
//SRTVCNTL DD *
* SORT BY VOLUME AND INDENTIFYING STATUS STRING
  SORT FIELDS=(1,6,CH,A,7,20,CH,A)
//SRTICNTL DD *
* SORT BY INDENTIFYING STATUS STRING AND VOLUME
  SORT FIELDS=(7,20,CH,A,1,6,CH,A)
//REPORTV DD SYSOUT=*
//REPORTI DD SYSOUT=*
//*
```

The first report is produced using ICETOOL's OCCUR operator. It shows the number of volumes in each of the three status classes. Here's what it might look like:

```
STATUS COUNT REPORT          09/05/96

STATUS                  NUMBER OF VOLUMES
--------------------    -----------------
IN CONVERSION TO SMS                    3
MANAGED BY SMS                         16
NON-SMS MANAGED                        10
```

The second report is produced using ICETOOL's DISPLAY operator. It shows the volumes in sorted order and the status of each. Here's what a portion of this report might look like:

```
VOLUME/STATUS REPORT         09/05/96          -1-

VOLUME    STATUS
------    --------------------
SHR200    NON-SMS MANAGED
SHR201    NON-SMS MANAGED
SHR202    NON-SMS MANAGED
.
.
.
SMS200    MANAGED BY SMS
SMS201    MANAGED BY SMS
.
.
.
SHR287    IN CONVERSION STATUS
.
.
.
```

The third report is another view of the data in the second report, obtained by switching the order of the fields. It shows the status classes in sorted order and each volume in that status class. Here's what a portion of this report might look like:

```
STATUS/VOLUME REPORT      09/05/96         -1-

STATUS              VOLUME
--------------------  ------
IN CONVERSION STATUS  SHR287
.
.
.
MANAGED BY SMS       SMS200
MANAGED BY SMS       SMS201
.
.
.
NON-SMS MANAGED      SHR200
NON-SMS MANAGED      SHR201
NON-SMS MANAGED      SHR202
.
.
.
```