What's New in DFSORT

October, 2010

Frank L. Yaeger

DFSORT Team
IBM Systems Software Development
San Jose, California
Internet: yaeger@us.ibm.com

DFSORT Web Site

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

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

Abstract

Over the years, DFSORT has provided important enhancements in many areas. This paper provides a quick introduction to selected DFSORT and ICETOOL enhancements available as of **October, 2010**. Examples using the various features are included, where appropriate.

Contents

What's New in DFSORT

Introduction

DFSORT is IBM's high performance sort, merge, copy, analysis and reporting product. DFSORT is an optional feature of z/OS.

DFSORT, together with DFSMS and RACF, form the strategic product base for the evolving system-managed storage environment. DFSMS provides vital storage and data management functions. RACF adds security functions. DFSORT adds the ability to do faster and easier sorting, merging, copying, reporting and analysis of your business information, as well as versatile data handling at the record, field and bit level.

Over the years, DFSORT has provided important enhancements for:

- resizing records
- updating counts and totals in trailer records
- date field arithmetic
- translation
- join and match operations
- date field conversions
- find and replace operations
- group operations
- sorting data between headers and trailers
- keeping or removing the first n records, last n records and/or specific relative records
- displaying and writing counts
- extracting variable position/length fields (e.g. CSV, delimited fields, keyword separated fields, etc) into fixed parsed fields
- justifying and squeezing data
- comparing and inserting past, current and future date constants
- testing for numerics and non-numerics
- displaying hexadecimal floating-point values as integers
- using SET, PROC and system symbols (e.g. &SYSPLEX) in control statements
- using symbols for output columns
- reformatting records before selecting or splicing
- conditionally reformatting records
- overlaying only selected parts of records
- larger numeric fields and constants
- new data formats
- restarting sequence numbers
- sampling, repeating and distributing records

- arithmetic operations using numeric fields and decimal constants
- reporting
- editing
- conversion
- date and time fields
- data analysis
- position and length limits
- productivity
- performance
- capacity
- storage usage
- input and output processing
- easier migration from other sort products.

This paper provides a quick introduction to selected DFSORT and ICETOOL enhancements available as of **October, 2010**. Examples using the various features are included, where appropriate.

Complete information on the newest features of DFSORT and DFSORT's ICETOOL can be found in *User Guide for DFSORT PTFs UK90025 and UK90026* (October, 2010).

Complete information on all of the features of DFSORT and DFSORT's ICETOOL through September, 2010 can be found in *z/OS DFSORT Application Programming Guide (SC26-7523-05)* and *z/OS DFSORT Installation and Customization (SC26-7524-03)*. If you're not familiar with DFSORT, DFSORT's ICETOOL or DFSORT Symbols, you should consider going through *z/OS DFSORT: Getting Started (SC26-7527-05)*. You can access these documents online by clicking the **Publications** link on the DFSORT home page at URL:

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

JOINKEYS Application

The new JOINKEYS application support is a significant addition to DFSORT's join and match capabilities. A JOINKEYS application makes it easy to create joined records in a variety of ways including inner join, full outer join, left outer join, right outer join, and unpaired combinations. The two input files can be of different types (fixed, variable, VSAM, and so on) and have keys in different locations.

Three new control statements can be used for a JOINKEYS application. One JOINKEYS statement is required for each input file to indicate the ddname of the file, describe the keys, indicate whether the file is already sorted by those keys, and so on. An inner join is performed by default, but a JOIN statement can be used to specify a different type of join. A REFORMAT statement is used to describe the fields from the two files to be included in the joined records, and optionally an indicator of where the key was found ('B' for both files, '1' for file1 only or '2' for file2 only).

The records from the two input files can be processed in a variety of ways before and after they are joined.

Here's an example of a Cartesian join of two files that have their keys in different locations.

```
//JK EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//VBIN DD DSN=...  VB input file
//FBIN DD DSN=...  FB input file
//SORTOUT DD DSN=...  FB output file
//SYSIN DD *
* Control statements for JOINKEYS application
  JOINKEYS F1=VBIN,FIELDS=(18,16,A),SORTED
  JOINKEYS F2=FBIN,FIELDS=(1,16,A)
  REFORMAT FIELDS=(F2:22,12,F1:5,12,F2:1,16)
* Control statements for main task (joined records)
  OPTION EQUALS
  SORT FIELDS=(13,12,CH,A)
/*
```

ICETOOL Features

DFSORT's popular multipurpose ICETOOL utility offers many important features:

- The **RESIZE operator** is a significant addition to ICETOOL's wide range of capabilities. For fixed-length records, RESIZE lets you create a larger record from multiple shorter records, or multiple shorter records from a larger record. You just indicate the desired length of the output records and RESIZE automatically resizes the records based on the input length.

  For example, the following ICETOOL job creates 40-byte output records from 10-byte input records:

  ```
  //INCRS EXEC  PGM=ICETOOL
  //TOOLMSG   DD  SYSOUT=*
  //DFSMSG    DD  SYSOUT=*
  //IN DD DSN=...   input file (FB/10)
  //OUT DD DSN=...  output file (FB/40)
  //TOOLIN DD *
  RESIZE FROM(IN) TO(OUT) TOLEN(40)
  /*
  ```

  If IN contained the following records:

  ```
  Bird
  Bluejay
  4
  Charlie
  Rodent
  Rat
  2
  Sara
  ```

  OUT would contain the following records:

  ```
  Bird      Bluejay   4         Charlie
  Rodent    Rat       2         Sara
  ```

- For the DISPLAY operator, the length for CH values has been raised from 1500 bytes to 4000 bytes, the length for HEX values has been raised from 1000 bytes to 2000 bytes, the number of ON fields has been raised from 20 to 50, and the line length has been raised from 2048 bytes to 8192 bytes. This lets you view and analyze more of your data.

- The **JKFROM operand** makes it easy to perform one or more JOINKEYS applications using ICETOOL's COPY or SORT operators.

- The **MERGE operator** lets you merge up to 50 input files that are already in order by the same keys. This makes it easy to include MERGE operations in your ICETOOL jobs.
- The **DATASORT operator** is a significant addition to ICETOOL's wide range of capabilities. DATASORT lets you sort the data records between header and trailer records while keeping the header and trailer records in place. You define the first n records as header records and/or the last n records as trailer records, and supply a DFSORT SORT statement for sorting the data records. DATASORT does the rest.

  For example, the following ICETOOL job sorts the data records on a CH field in positions 5-10 while keeping the first 2 records (header records) and last record (trailer record) in place.

  ```
  //DTASRT EXEC  PGM=ICETOOL
  //TOOLMSG   DD  SYSOUT=*
  //DFSMSG    DD  SYSOUT=*
  //IN DD DSN=...   input file
  //OUT DD DSN=...  output file
  //TOOLIN DD *
  DATASORT FROM(IN) TO(OUT) HEADER(2) TRAILER USING(CTL1)
  /*
  //CTL1CNTL DD *
    SORT FIELDS=(5,6,CH,A)
  /*
  ```
- The **SUBSET operator** makes it easy to create an output data set with a subset of records from an input data set by keeping or removing the first n records (headers), specified relative records, and/or the last n records (trailers). You tell SUBSET whether you want to keep or remove input or output records, and which header, relative or trailer records you want to select to be kept or removed. You can even direct the records that are not selected to a separate data set.

  For example, the following ICETOOL job keeps the first input record, relative records 11-15 and 21, and the last 2 records.

  ```
  //SBSET EXEC  PGM=ICETOOL
  //TOOLMSG   DD  SYSOUT=*
  //DFSMSG    DD  SYSOUT=*
  //IN DD DSN=...   input file
  //OUT DD DSN=...  output file
  //TOOLIN DD *
  SUBSET FROM(IN) TO(OUT) FIRST RRN(11,15) RRN(21) LAST(2)
  /*
  ```
- For the **SELECT operator, FIRST(n)** and **FIRSTDUP(n)** let you select the first n records with each key or the first n duplicate records with each key, respectively. FIRST(n) and FIRSTDUP(n) join FIRST, FIRSTDUP, LAST, LASTDUP, LAST, ALLDUPS, NODUPS, HIGHER(n), LOWER(n) and EQUAL(n) in SELECT's bag of tricks for dealing with duplicates. FIRST(n) and FIRSTDUP(n) make it easy to pick the top n or bottom n values with each key.

  In the following example using FIRST(n), the top 3 values (that is, the highest values) with each key are written to OUTPUT1.

```
//SELN EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//INPUT DD DSN=...   input file
//OUTPUT DD DSN=...  output file
//TOOLIN DD *
SELECT FROM(INPUT) TO(OUTPUT) ON(1,5,CH) FIRST(3) USING(CTL1)
/*
//CTL1CNTL DD *
  SORT FIELDS=(1,5,CH,A,11,3,ZD,D)
/*
```

If INPUT contained the following records:

```
JUNE      025
APRIL     077
JUNE      072
JUNE      124
APRIL     051
APRIL     058
JUNE      082
JUNE      095
APRIL     005
```

OUTPUT would contain the following records:

```
APRIL     077
APRIL     058
APRIL     051
JUNE      124
JUNE      095
JUNE      082
```

USING(xxxx) can be used to supply DFSORT control statements INCLUDE, OMIT, INREC, OPTION, SORT and OUTFIL for SELECT processing. INCLUDE and OMIT can be used to delete records that you don't want to SELECT. INREC can be used to reformat the input records before they are selected. OPTION can be used to specify optional parameters such as MAINSIZE. SORT can be used to specify fields you want to sort on, but not select on. OUTFIL statements can be used to further process the selected records.

In the following example, an OMIT statement is used to ensure that records with C'NO' in positions 25-26 are not processed by the SELECT operator, and a SORT statement is used to sort/select the records ascending by the field in positions 12-16 and sort the records descending by the field in positions 35-40.

```
  SELECT FROM(IN) TO(OUT) ON(12,5,CH) FIRSTDUP USING(CTL1)
//CTL1CNTL DD *
  OMIT COND=(25,2,CH,EQ,C'NO')
  SORT FIELDS=(12,5,CH,A,35,6,CH,D)
```

In the following example, OUTFIL statements are used to reformat selected and discarded records for certain record types.

```
  SELECT FROM(INPUT1) TO(OUT1) DISCARD(OUT2) -
    ON(31,6,ZD) FIRST USING(CTL2)
//CTL2CNTL DD *
  OUTFIL FNAMES=OUT1,INCLUDE=(6,1,BI,EQ,+1,OR,6,1,BI,EQ,+5),
   BUILD=(1,20,21,4,PD,TO=ZD,90:X)
  OUTFIL FNAMES=OUT2,INCLUDE=(6,1,BI,EQ,+12),
   BUILD=(1,12,13,5,PD,TO=ZD,X,42,4,FI,TO=ZD,120:X)
```

FIRSTDUP and **LASTDUP** let you select the first or last record of each set of duplicates.

Additionally, **DISCARD(savedd)** lets you keep the records that do not meet your SELECT criteria at the same time you use TO(indd) to keep the records that do meet your criteria. You can use TO and DISCARD together or separately.

In the following example using FIRSTDUP and DISCARD, the first record of each set of duplicates is written to OUTPUT1. All other records (non-duplicates, second and subsequent duplicates) are written to OUTPUT2.

```
   SELECT FROM(INPUT) TO(OUTPUT1) ON(1,3,CH) FIRSTDUP -
     DISCARD(OUTPUT2)
```

If INPUT contained the following records:

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

OUTPUT1 would contain the following records:

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

and OUTPUT2 would contain the following records:

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

- The **SPLICE operator** lets you create output records in a variety of ways by splicing together fields from records that have the same key, but different information. The fields to be spliced can originate from records in different data sets, so you an use SPLICE to do various "join" and "match" operations.

  The records to be spliced must have their keys in the same location. FB records to be spliced must also be the same length. So typically, you will want to reformat one or more input data sets and concatenate them for input to SPLICE.

  KEEPBASE, KEEPNODUPS, WITHALL, WITHANY and WITHEACH can be used to control which records are kept and spliced.

  VLENOVLY and **VLENMAX** can be used to set the record length for spliced VB records.

  USING(xxxx) can be used to supply DFSORT control statements INCLUDE, OMIT, INREC, OPTION and OUTFIL for SPLICE processing. INCLUDE and OMIT can be used to delete records that you don't want in the output data set. INREC can be used to reformat the input records before they are spliced. OPTION can be used to specify optional parameters such as MAINSIZE. OUTFIL statements can be used to further process the spliced records.

  In the following SPLICE example, a 3-digit city code is used to join information from two different input data sets.

  IN1 has RECFM=FB and LRECL=30. It contains the following records:

```
503 San Jose  CA  1967-12-24
207 New York  NY  1992-05-18
420 Boston    MA  1986-09-21
806 Denver    CO  2001-10-03
721 Chicago   IL  1975-02-08
```

  IN2 has RECFM=FB and LRECL=20. It contains the following records:

```
Vezinaw      207
Terradista   503
Van Noorden  207
Yaeger       503
Paulsen      806
Samuels      806
Wilson       207
```

We want to join the records with matching city codes in positions 1-3 of IN1 and positions 14-16 of IN2 to produce an output data set with RECFM=FB and LRECL=60 that contains the following records:

```
Vezinaw      207 New York  NY  1992-05-18
Van Noorden  207 New York  NY  1992-05-18
Wilson       207 New York  NY  1992-05-18
Terradista   503 San Jose  CA  1967-12-24
Yaeger       503 San Jose  CA  1967-12-24
Paulsen      806 Denver    CO  2001-10-03
Samuels      806 Denver    CO  2001-10-03
```

The following ICETOOL job uses the SPLICE operator to join the records.

```
//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=(CYL,(5,5)),DISP=(MOD,PASS)
//OUT DD DSN=...  output file
//TOOLIN DD *
* Reformat IN1 to contain:
* | blanks   | key | IN1 data |
  COPY FROM(IN1) TO(T1) USING(CTL1)
* Reformat IN2 to contain:
* | IN2 data | key | blanks   |
  COPY FROM(IN2) TO(T1) USING(CTL2)
* SPLICE the matching IN1/IN2 records to produce:
* | IN2 data | key | IN1 data |
  SPLICE FROM(T1) TO(OUT) ON(14,3,CH) -
    WITHALL WITH(1,13)
/*
//CTL1CNTL DD *
  OUTREC BUILD=(14:1,3,17:4,26,60:X)
/*
//CTL2CNTL DD *
  OUTREC BUILD=(1,20,60:X)
/*
```

In the next SPLICE example, the input records from data sets FILE1 and FILE2 are separated into the following output data sets:

- BOTH: records that appear in FILE1 and FILE2
- F1ONLY: records that only appear in FILE1
- F2ONLY: records that only appear in FILE2

Here's the ICETOOL job:

```
//S1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//FILE1 DD *
Vicky
Frank
Carrie
Holly
Paul
/*
//FILE2 DD *
Karen
Holly
Carrie
Vicky
Mary
/*
//BOTH  DD SYSOUT=*   output for records in File1 and File2
//F1ONLY DD SYSOUT=*  output for records in File1 only
//F2ONLY DD SYSOUT=*  output for records in File2 only
//T1 DD DSN=&&T1,DISP=(MOD,PASS),UNIT=SYSDA,SPACE=(TRK,(5,5))
//TOOLIN DD *
* Add '11' identifier for FILE1 records.
COPY FROM(FILE1) TO(T1) USING(CTL1)
* Add '22' identifier for FILE2 records.
COPY FROM(FILE2) TO(T1) USING(CTL2)
* SPLICE to match up records and write them to their
* appropriate output files.
*   BOTH records will have an identifier of '12'
*   F1ONLY records will have an identifier of '11'
*   F2ONLY records will have an identifier of '22'
SPLICE FROM(T1) TO(BOTH) ON(1,10,CH) WITH(13,1) -
USING(CTL3) KEEPNODUPS
/*
//CTL1CNTL DD *
* Mark FILE1 records with '11'
  OUTREC BUILD=(1,10,12:C'11')
/*
//CTL2CNTL DD *
* Mark FILE2 records with '22'
  OUTREC BUILD=(1,10,12:C'22')
/*
//CTL3CNTL DD *
* Write matching records to BOTH file. Remove id.
  OUTFIL FNAMES=BOTH,INCLUDE=(12,2,CH,EQ,C'12'),BUILD=(1,10)
* Write FILE1 only records to F1ONLY file. Remove id.
  OUTFIL FNAMES=F1ONLY,INCLUDE=(12,2,CH,EQ,C'11'),BUILD=(1,10)
* Write FILE2 only records to F2ONLY file. Remove id.
  OUTFIL FNAMES=F2ONLY,INCLUDE=(12,2,CH,EQ,C'22'),BUILD=(1,10)
/*
```

BOTH contains the following records:

```
Carrie
Holly
Vicky
```

F1ONLY contains the following records:

```
Frank
Paul
```

F2ONLY contains the following records:

```
Karen
Mary
```

- For the **COUNT operator, WRITE(countdd)** lets you write the count of records to an output data set. **TEXT('string')** can be used to insert a text string before the count of records, and **DIGITS(d)** or **EDCOUNT(formatting)** can be used to format the count of records.

  For example, the following COUNT operator writes a record in CT1 with text and the count of records from IN1:

  ```
  COUNT FROM(IN1) WRITE(CT1) TEXT('Record count is ') -
   EDCOUNT(A1,U08)
  ```

  If IN1 contained 865234 records, CT1 would contain this record:

  ```
  Record count is     865,234
  ```

  SUB(n) can be used to subtract n from the count of records and **ADD(n)** can be used to add n to the count of records. This is especially useful for dealing with data sets that contain header and/or trailer records.

  For example, if IN2 has a header, data records and a trailer record, the following COUNT operator writes a record in CT2 with the count of data records:

  ```
  COUNT FROM(IN2) WRITE(CT2) SUB(2) DIGITS(3)
  ```

  If IN2 contained 375 records, CT2 would contain this record:

  ```
  373
  ```

  A return code of 12, 8 or 4 can be set 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, 50000). This makes it easy to control the execution of downstream operators or steps using JCL facilities like IF or COND. A return code of 12 is set with **RC12** or by default. **RC8** can be used to set a return code of 8 instead of a return code of 12. **RC4** can be used to set a return code of 4 instead of a return code of 12.

  For example, in the following job, the COUNT operator stops STEP2 from being executed by setting a return code of 12 if the IN data set is empty.

  ```
  //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
  ```

- The **DISPLAY and OCCUR operators** create useful reports with a minimum of work. Various options let you create even better looking reports.

TITLE('part1','part2','part3') can be used to specify up to three title lines with up to three different strings or symbols for each title line. **TLEFT** can be used to left-justify the title lines (instead of centering them). **TFIRST** can be used to display the title lines on the first page only (instead of on every page).

NOCC can be used to create reports without carriage control characters.

DATE, DATE(abcd), DATENS(abc), YDDD(abc) and **YDDDNS(ab)** can be used to generate current date constants in various forms, with or without separators, where a, b, c and d represent various parts of the date (mm, dd, yy, yyyy, ddd) and various separators (for example, ., - or /). **TIME, TIME(abc)** and **TIMENS(ab)** can be used to generate current time constants in various forms, with or without separators, where ab represents 12-hour or 24-hour time and c represents various separators (for example, :, . or -).

HEADER('string1'), HEADER('string1','string2') and **HEADER('string1','string2','string3')** can be used to create one, two or three line headings, respectively, for data columns.

INDENT(n) can be used to indent the report by n spaces. **BETWEEN(n)** can be used to put n spaces between the data columns. **TBETWEEN(n)** can be used to put n spaces between the title elements. **STATLEFT** can be used to place statistics strings to the left of the first column of data.

Additional formatting items such as **G1-G6** (show numeric values with 4 decimal places), **E'pattern'** (use a specified pattern for numeric digits such as dates, phone numbers, and so on), **LZ** (insert leading zeros), **NOST** (suppress statistics), **Udd** (use dd digits), **/D** (divide by 10) and **/C** (divide by 100) join the existing formatting items to give you new ways to present your data in reports.

Formatting items can be used for **VLEN, NUM, VALCNT** and **BREAK** values as well as for **ON** values.

Here's an example of a DISPLAY operator that uses some of the new options to improve the appearance of a report:

```
 DISPLAY FROM(ACCTS) LIST(FANCY) -
    DATE(MD4/) -
    TITLE('Accounts Report for First Quarter') -
    TITLE('with Total and Average') -
    BLANK TBETWEEN(16) -
    HEADER(,'Amount') ON(12,6,ZD,C1,U08) -
    HEADER(,'Id') ON(NUM,U02) -
    HEADER('Account',' Number') ON(31,3,PD,NOST,LZ) -
    HEADER(,'Date') ON(1,4,ZD,E'99/99',NOST) -
    INDENT(2) BETWEEN(5) -
    STATLEFT -
    TOTAL('Total for Q1') -
    AVERAGE('Average for Q1')
```

The FANCY report might look as follows:

```
06/17/2008              Accounts Report for First Quarter

                                 with Total and Average

                                                   Account
                         Amount         Id          Number         Date
                    -----------        ---         -------        -----
                         932.71          1           15932        01/06
                       1,376.22          2           00187        01/28
                         831.47          3           15932        02/12
                       1,832.61          4           02158        02/17
                         763.89          5           00187        03/05
                       9,200.13          6           15932        03/19

Total for Q1          14,937.03

Average for Q1         2,489.50
```

Various formats let you edit SMF, TOD (STCK) and ETOD (STCKE) date and time values into more recognizable forms as follows:

Format	Result
DT1	SMF date interpreted as Z'yyyymmdd'
DT2	SMF date interpreted as Z'yyyymm'
DT3	SMF date interpreted as Z'yyyyddd'
DC1	TOD date interpreted as Z'yyyymmdd'
DC2	TOD date interpreted as Z'yyyymm'
DC3	TOD date interpreted as Z'yyyyddd'
DE1	ETOD date interpreted as Z'yyyymmdd'
DE2	ETOD date interpreted as Z'yyyymm'
DE3	ETOD date interpreted as Z'yyyyddd'
TM1	SMF time interpreted as Z'hhmmss'
TM2	SMF time interpreted as Z'hhmm'
TM3	SMF time interpreted as Z'hh'
TM4	SMF time interpreted as Z'hhmmssxx'
TC1	TOD time interpreted as Z'hhmmss'
TC2	TOD time interpreted as Z'hhmm'
TC3	TOD time interpreted as Z'hh'
TC4	TOD time interpreted as Z'hhmmssxx'
TE1	ETOD time interpreted as Z'hhmmss'
TE2	ETOD time interpreted as Z'hhmm'
TE3	ETOD time interpreted as Z'hh'
TE4	ETOD time interpreted as Z'hhmmssxx'

The interpreted values can be further edited using formatting items. This makes it easy to show SMF, TOD and ETOD date and time values in meaningful ways.

The following example shows how SMF date and time values can be displayed as easily understood data in a report on SMF type-14 records.

```
  DISPLAY FROM(SMF14) LIST(SMF14RPT) -
    TITLE('SMF Type-14 Records') DATE(4MD/) -
    HEADER('Date') ON(11,4,DT1,E'9999/99/99') -
    HEADER('Time') ON(7,4,TM1,E'99:99:99') -
    HEADER('Sys') ON(15,4,CH) -
    HEADER('Jobname') ON(19,8,CH) -
    HEADER('Datasetname') ON(69,44,CH)
```

The SMF14RPT report might look as follows:

```
 SMF Type-14 Records        2005/03/15

       Date       Time   Sys    Jobname    Datasetname
 ----------   --------   ----   --------   ------------- ...
 2005/02/20   06:03:15   ID03   JOB00003   SYS1.QRS
 2005/02/20   10:03:22   ID02   JOB00002   SYS1.XYZ
 2005/02/21   14:05:37   ID03   JOB00004   SYS1.MNO
 2005/02/21   22:11:00   ID03   JOB00005   SYS1.MNO
 2005/02/24   00:00:08   ID03   JOB00006   SYS1.MNO
```

- For the **DISPLAY operator, COUNT('string')** and **BCOUNT('string')** lets you display an overall record count and break record counts, respectively similar to the way you can display other statistics (maximum, minimum, total and average). **EDCOUNT(formatting)** can be used to format the overall count. **EDBCOUNT(formatting)** can be used to format the break count.

  Here's an example of a DISPLAY operator that displays an overall count:

```
DISPLAY FROM(MASTER) LIST(RPT1) -
  HEADER('City') ON(1,15,CH) -
  COUNT('Number of cities:') EDCOUNT(U03)
```

  The RPT1 report might look as follows:

```
City
--------------------
Gilroy
Morgan Hill
San Martin
Los Gatos

Number of cities:   4
```

  FL format lets you display signed hexadecimal floating point values, totals, minimums, maximums and averages as their corresponding signed integer values. The integer values can be further edited using formatting items. This makes it easy to display hexadecimal floating point values and statistics as integers.

  Here's an example of displaying hexadecimal floating point values with DISPLAY:

```
DISPLAY FROM(SMF71) LIST(SMF71RPT) -
  TITLE('Low impact central storage frames') -
  HEADER('Min Frames') ON(925,8,FL,U10) -
  HEADER('Max Frames') ON(933,8,FL,U10) -
  HEADER('Avg Frames') ON(941,8,FL,U10) -
  BLANK PAGE
```

Changing Installation Options via PARMLIB

ICEPRMxx members in concatenated PARMLIB can now be used to specify changes to DFSORT s installation options. Each ICEPRMxx member can contain options to be changed for any or all of DFSORT s eight installation environments (JCL, INV, TSO, TSOINV and TD1-TD4). Up to ten ICEPRMxx members can be activated by a START ICEOPT started task command. The options in the activated members will be merged with the ICEMAC defaults at run-time.

A different ICEPRMxx member, or combination of ICEPRMxx members, for different LPARs can be activated at IPL time by including a START ICEOPT command in an appropriate COMMNDxx member in PARMLIB, or at any time by issuing a START ICEOPT command from the console.

ICEPRMxx members are are easier to use and more flexible then the old method using the ICEMAC macro and usermods.

The ICETOOL DEFAULTS operator can be used at any time to produce a report showing the merged PARMLIB/ICEMAC installation default values for each environment that will be used at run-time, as well as the active ICEPRMxx and ICEMAC values.

Reformatting Features (INREC, OUTREC, OUTFIL)

Many reformatting features are available with DFSORT's INREC, OUTREC and OUTFIL statements, making each useful by itself or in conjunction with the other statements.

- INREC, OUTREC and OUTFIL can translate data in several new ways as follows:
  - **TRAN=ATOE** translates ASCII characters anywhere in a specified field to their equivalent EBCDIC characters. As a simple example, if you specify:

    ```
    INREC OVERLAY=(11:11,3,TRAN=ATOE)
    ```

    DFSORT will translate ASCII to EBCDIC in positions 11-13 of each output record. So ASCII aB2 (X'614232') would be translated to EBCDIC aB2 (X'81C2F2').
  - **TRAN=ETOA** translates EBCDIC characters anywhere in a specified field to their equivalent ASCII characters. As a simple example, if you specify:

    ```
    INREC OVERLAY=(21:21,3,TRAN=ETOA)
    ```

    DFSORT will translate EBCDIC to ASCII in positions 21-23 of each output record. So EBCDIC aB2 (X'81C2F2') would be translated to ASCII aB2 (X'614232').
  - **TRAN=HEX** translates binary values anywhere in a specified field to their equivalent EBCDIC hexadecimal characters. As a simple example, if you specify:

    ```
    INREC OVERLAY=(5:51,2,TRAN=HEX)
    ```

    DFSORT will translate binary to hex in positions 51-52 of each output record. So X'C1F1' (C'A1') would be translated to C'C1F1'.
  - **TRAN=UNHEX** translates EBCDIC hexadecimal characters anywhere in a specified field to their equivalent binary values. As a simple example, if you specify:

    ```
    INREC BUILD=(1,4,TRAN=UNHEX)
    ```

    DFSORT will translate hex to binary in positions 1-4 of each output record. So C'C1F1 would be translated to X'C1F1' (C'A1').

- **TRAN=BIT** translates binary values anywhere in a specified field to their equivalent EBCDIC bit values. As a simple example, if you specify:

  ```
  INREC OVERLAY=(16:16,2,TRAN=BIT)
  ```

  DFSORT will translate binary to bits in positions 16-17 of each output record. So X'C1F1' (C'A1') would be translated to C'1100000111110001'.

- **TRAN=UNBIT** translates EBCDIC bit values anywhere in a specified field to their equivalent binary values. As a simple example, if you specify:

  ```
  INREC BUILD=(31:31,16,TRAN=UNBIT)
  ```

  DFSORT will translate bit to binary in positions 31-46 of each output record. So C'1100000111110001' would be translated to X'C1F1' (C'A1').

- **TRAN=LTOU** translates lowercase EBCDIC letters anywhere in a specified field to uppercase EBCDIC letters. As a simple example, if you specify:

  ```
  INREC OVERLAY=(31:31,11,TRAN=LTOU)
  ```

  DFSORT will translate lowercase to uppercase in positions 31-41 of each output record. So 'Vicky-123,x' would be translated to 'VICKY-123,X'.

- **TRAN=UTOL** translates uppercase EBCDIC letters anywhere in a specified field to lowercase EBCDIC letters. As a simple example, if you specify:

  ```
  OUTFIL BUILD=(1,4,5,TRAN=UTOL)
  ```

  DFSORT will translate uppercase to lowercase in the entire data portion of each variable-length output record. So 'CARRIE-005, CA' would be translated to 'carrie-005, ca'.

- INREC, OUTREC and OUTFIL allow you to perform various types of arithmetic on date fields in either Julian or Gregorian form, with 2-digit or 4-digit years, in CH, ZD and PD format.

  ADDDAYS, ADDMONS and ADDYEARS can be used to add days, months or years to a date field.

  SUBDAYS, SUBMONS and SUBYEARS can be used to subtract days, months or years from a date field.

  DATEDIFF can be used to calculate the number of days between two date fields.

  NEXTDday can be used to calculate the next specified day of the week for a date field.

  PREVDday can be used to calculate the previous specified day of the week for a date field.

  LASTDAYW, LASTDAYM, LASTDAYQ and LASTDAYY can be used to calculate the last day of the week, month, quarter or year for a date field.

  Here's an example of adding 50 days, subtracting 7 months and adding 2 years to a ccyymmdd date:

  ```
  INREC BUILD=(11:1,8,Y4T,ADDDAYS,+50,TOGREG=Y4T,
               21:1,8,Y4T,SUBMONS,+7,TOGREG=Y4T,
               31:1,8,Y4T,ADDYEARS,+2,TOGREG=Y4T)
  ```

- INREC, OUTREC and OUTFIL can use the **Y2x, Y4x, TOJUL, TOGREG, WEEKDAY, DT and DTNS keywords** to convert input data fields of one type to corresponding output date fields of another type, and to convert a date field to a corresponding day of the week in several forms.

  The date field can be in either Julian or Gregorian form, with 2-digit or 4-digit years, in CH, ZD and PD format. For example, C'ccyyddd', P'yymmdd', Z'dddccyy', C'ccyymmdd', P'dddyy', and so on. CH output date fields can also be displayed with separators. For example, C'ccyy/ddd', C'mm-dd-ccyy', and so on.

  The day of the week can be displayed as a 1 digit, 3 character or 9 character constant. For example, C'4', C'WED' or C'WEDNESDAY' for Wednesday.

Here's an example of converting an mmddyy date to a ccyyddd date and a 3-character weekday string, and converting a ccyyddd date to a ccyy/mm/dd date and 1-digit weekday string.

```
OPTION COPY,Y2PAST=1996
INREC BUILD=(1,6,Y2W,TOJUL=Y4T,X,
    1,6,Y2W,WEEKDAY=CHAR3,X,
    9,7,Y4T,TOGREG=Y4T(/),X,
    9,7,Y4T,WEEKDAY=DIGIT1)
```

- INREC, OUTREC and OUTFIL can create reformatted records in one of the following ways using unedited, edited, or converted input fields and a variety of constants, as appropriate:
  - **FINDREP**: Reformat each record by replacing character or hexadecimal input constants anywhere in the record with character, hexadecimal or null output constants. FINDREP lets you do a variety of find and replace operations on your records.

    Here's an example of FINDREP with OUTREC:

    ```
    OUTREC FINDREP=(IN=(C'D27',C'A52',C'X31'),OUT=C'INVALID')
    ```

  - **BUILD**: Reformat each record by specifying all of its item one by one. BUILD gives you complete control over the items you want in your reformatted INREC records and the order in which they appear. You can delete, rearrange and insert fields and constants.

    Here's an example of BUILD with INREC:

    ```
    INREC BUILD=(1,20,C'ABC',26:5C'*',
           15,3,PD,EDIT=(TTT.TT),21,30,80:X)
    ```

    Note: For INREC or OUTREC, you can use **FIELDS** or **BUILD**. For OUTFIL, you can use **OUTREC** or **BUILD**.

  - **OVERLAY**: Reformat each record by specifying just the items that overlay specific columns. OVERLAY lets you change specific existing columns without affecting the entire record.

    Here's an example of OVERLAY with OUTREC:

    ```
    INREC OVERLAY=(45:45,8,TRAN=LTOU)
    ```

    Lowercase letters in positions 45-52 of the input record will be changed to uppercase letters in positions 45-52 of the output records. All other bytes in the input record will be copied to the output record without change.

  - **IFTHEN clauses**: Reformat different records in different ways by specifying how FINDREP, BUILD or OVERLAY items are applied to records that meet given criteria. IFTHEN clauses let you use sophisticated conditional logic to choose how different record types are reformatted.

    Here's an example of IFTHEN clauses with OUTFIL:

    ```
    OUTFIL IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE1'),
             BUILD=(1,40,C'**',+1,TO=PD)),
           IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE2'),
             BUILD=(1,40,+2,TO=PD,X'FFFF')),
           IFTHEN=(WHEN=NONE),OVERLAY=(45:C'NONE'))
    ```

    IFTHEN also lets you do various types of operations involving groups of records by making it easy to propagate fields from the first record of a group to the other records of the group, or add an identifier and/or sequence number to each record of the group. These functions are useful by themselves, and can also facilitate other types of group operations such as sorting by groups, including or omitting groups, and so on.

    Here's an example of including groups of records consisting of a HDR record, data records and a TRL record, while excluding records before, between and after the groups.

```
OPTION COPY
INREC IFTHEN=(WHEN=GROUP,BEGIN=(1,3,CH,EQ,C'HDR'),
  END=(1,3,CH,EQ,C'TRL'),PUSH=(31:ID=1))
OUTFIL INCLUDE=(31,1,CH,NE,C' '),BUILD=(1,30)
```

- INREC, OUTREC and OUTFIL can use the **JFY** parameter to left-justify or right-justify the data in a field. For left-justified data, blanks on the left are removed and the remaining data is shifted left. For right-justified data, blanks on the right are removed and the remaining data is shifted right.

  Optionally, while justifying, specific leading and trailing characters can be treated as blanks, leading and trailing characters can be inserted, and the output length can be changed.

  Here's an example of left-justifying and right-justifying:

  ```
  INREC OVERLAY=(21:21,18,JFY=(SHIFT=LEFT,
           LEAD=C'<',TRAIL=C'>',LENGTH=20),
                 1:1,12,JFY=(SHIFT=RIGHT),
                 501:51,9,JFY=(SHIFT=LEFT,PREBLANK=C'()'))
  ```

- INREC, OUTREC and OUTFIL can use the **SQZ** parameter to left-squeeze or right-squeeze the data in a field. For left-squeezed data, blanks on the left and in the middle are removed and the remaining data is shifted left. For right-squeezed data, blanks on the right and in the middle are removed and the remaining data is shifted right.

  Optionally, while squeezing, specific leading and trailing characters can be treated as blanks, leading, middle and trailing characters can be inserted, blanks can be kept between apostrophes or quotes, and the output length can be changed.

  Here's an example of left-squeezing and right-squeezing:

  ```
  OUTREC BUILD=(1,20,
    21,15,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',',LENGTH=20),
    58,12,SQZ=(SHIFT=RIGHT,LEAD=C'$',PAIR=QUOTE))
  ```

- INREC, OUTREC and OUTFIL can use the **PARSE** parameter to extract variable position/length delimited fields into **%nn** parsed fields. The %nn fields can then be used where p,m fields can be used. PARSE allows you to define the rules for extracting variable fields, such as delimited fields, comma separated values (CSV), tab separated values, keyword separated fields, null-terminated strings, and so on, into up to one hundred %nn fixed-length parsed fields (%00-%99). You can edit, convert, justify, squeeze, translate and do arithmetic with %nn fields. You can use % fields to skip variable values you don't care about.

  Here's an example of extracting the first, third and fourth comma separated values into unedited and edited fixed fields:

  ```
  OUTREC PARSE=(%01=(ENDBEFR=C',',FIXLEN=10),
                %=(ENDBEFR=C','),
                %03=(ENDBEFR=C',',FIXLEN=8),
                %04=(ENDBEFR=C',',FIXLEN=12)),
         BUILD=(5:%01,
                21:%03,SFF,TO=FS,LENGTH=10,
                41:%04,TRAN=UTOL)
  ```

- INREC, OUTREC and OUTFIL can generate current date constants in various forms, with or without separators, using the parameters **DATE=(abcd), DATENS=(abc), YDDD=(abc)** and **YDDDNS=(ab)** where a, b, c and d represent various parts of the date (mm, dd, yy, yyyy, ddd) and various separators (for example, ., - or /). INREC, OUTREC and OUTFIL can generate current time constants in various forms, with or without separators, using **TIME=(abc)** and **TIMENS=(ab)** where ab represents 12-hour or 24-hour time and c represents various separators (for example, :, . or -).

  INREC, OUTREC and OUTFIL can also generate current, past and future date constants, and current time constants as follows (d is days, m is months and c is any character except a blank):

Operand	**Constant**
DATE	C'mm/dd/yy'
DATE1, DATE1-d, DATE1+d	C'yyyymmdd'
DATE1(c), DATE1(c)-d, DATE1(c)+d	C'yyyycmmcdd'
DATE1P, DATE1P-d, DATE1P+d	P'yyyymmdd'
DATE2, DATE2-m, DATE2+m	C'yyyymm'
DATE2(c), DATE2(c)-m, DATE2(c)+m	C'yyyycmm'
DATE2P, DATE2P-m, DATE2P+m	P'yyyymm'
DATE3, DATE3-d, DATE3+d	C'yyyyddd'
DATE3(c), DATE3(c)-d, DATE3(c)+d	C'yyyycddd'
DATE3P, DATE3P-d, DATE3P+d	P'yyyyddd'
DATE4	C'yyyy-mm-dd-hh.mm.ss'
DATE5	C'yyyy-mm-dd-hh.mm.ss.nnnnnn'
TIME	C'hh:mm:ss'
TIME1	C'hhmmss'
TIME1(c)	C'hhcmmcss'
TIME1P	P'hhmmss'
TIME2	C'hhmm'
TIME2(c)	C'hhcmm'
TIME2P	P'hhmm'
TIME3	C'hh'
TIME3P	P'hh'

This makes it easy to insert timestamps of various types into your output records. For example, if you used the following OUTREC statement for a DFSORT run on March 16, 2005 at 03:46:21pm:

```
OUTREC BUILD=(2X,YDDD=(D4.),X,TIME1(:),X,1,10)
```

your SORTOUT records would look as follows:

```
075.2005 15:46:21 data
```

- INREC, OUTREC and OUTFIL can convert signed hexadecimal floating point values (FL) into their corresponding signed integer values. The integer values can be further edited using edit masks or edit patterns, or further converted to BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or FS/CSF values. This makes it easy to display and do arithmetic on hexadecimal floating point values.

  Here's an example of displaying a hexadecimal floating point value as an integer using an edit mask:

  ```
   INREC BUILD=(1,40,41:51,4,FL,M12)
  ```

- INREC, OUTREC and OUTFIL can display a numeric field in various ways using DFSORT's **M0-M26** predefined edit masks, or edit patterns you define yourself using **EDIT=(pattern)** and **SIGNS=(signs)**. You can edit BI, FI, PD, PD0, ZD, CSF/FS, UFF, SFF, FL or Y2x input fields using edit masks or edit patterns. The length of the output field can be defaulted or specified. As a simple example, if you specify:

```
  OUTREC BUILD=(21,6,ZD,EDIT=(SII,IIT.T),SIGNS=(+,-),X,
   6,5,PD,M7)
```

a zoned decimal value of 053214 in positions 21-26 of the input record will be displayed as ' +5,321.4' in positions 1-9 of the output record, and a packed decimal value of 235107283 in positions 6-10 of the input record will be displayed as '235-10-7283' in positions 11-21 of the output record.

- INREC, OUTREC and OUTFIL can convert a field in one numeric format to another numeric format. You can convert BI, FI, PD, PD0, ZD, CSF/FS, UFF, SFF, FL or Y2x input fields to BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or CSF/FS output fields. The length of the output field can be defaulted or specified. As a simple example, if you specify:

```
  OUTREC BUILD=(21,5,ZD,TO=PD,8,4,ZD,TO=FI,LENGTH=2)
```

  the zoned decimal values in positions 21-25 and 8-11 of the input records will be converted, respectively, to a packed decimal value in positions 1-3 and a fixed-point value in positions 4-5 of the output records.

- INREC, OUTREC and OUTFIL can edit SMF, TOD (STCK) and ETOD (STCKE) date and time values into more recognizable forms as follows:

Format	**Result**
DT1	SMF date interpreted as Z'yyyymmdd'
DT2	SMF date interpreted as Z'yyyymm'
DT3	SMF date interpreted as Z'yyyyddd'
DC1	TOD date interpreted as Z'yyyymmdd'
DC2	TOD date interpreted as Z'yyyymm'
DC3	TOD date interpreted as Z'yyyyddd'
DE1	ETOD date interpreted as Z'yyyymmdd'
DE2	ETOD date interpreted as Z'yyyymm'
DE3	ETOD date interpreted as Z'yyyyddd'
TM1	SMF time interpreted as Z'hhmmss'
TM2	SMF time interpreted as Z'hhmm'
TM3	SMF time interpreted as Z'hh'
TM4	SMF time interpreted as Z'hhmmssxx'
TC1	TOD time interpreted as Z'hhmmss'
TC2	TOD time interpreted as Z'hhmm'
TC3	TOD time interpreted as Z'hh'
TC4	TOD time interpreted as Z'hhmmssxx'
TE1	ETOD time interpreted as Z'hhmmss'
TE2	ETOD time interpreted as Z'hhmm'
TE3	ETOD time interpreted as Z'hh'
TE4	ETOD time interpreted as Z'hhmmssxx'

The interpreted values can be further edited using edit masks or edit patterns, or converted to BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or FS/CSF values. This makes it easy to show SMF, TOD and ETOD date and time values in meaningful ways.

The following example shows how TOD date and time values can be converted to readable form:

```
* Display a TOD date as C'yyyy/mm/dd' and a
* TOD time as 'hh:mm:ss.xx'.
  OUTREC BUILD=(X,26,8,DC1,EDIT=(TTTT/TT/TT),X,
    26,8,TC4,EDIT=(TT:TT:TT.TT))
```

The SORTOUT output might look as follows:

```
 2005/02/09 10:27:04
 2005/02/10 06:13:21
 2005/03/05 12:07:33
 2005/03/22 06:43:08
```

- INREC, OUTREC and OUTFIL can combine numeric fields (p,m,f), decimal constants (+n and -n), operators (**MIN, MAX, MUL, DIV, MOD, ADD, SUB)** and parentheses to form arithmetic expressions. The results can be further edited using editing masks or editing patterns, or converted to BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or FS/CSF values.

  Here's an example with an arithmetic expression:

  ```
    OUTFIL FNAMES=OUT,
      BUILD=(5:C'% REDUCTION FOR ',21,8,C' IS ',
        ((11,6,ZD,SUB,31,6,ZD),MUL,+1000),DIV,11,6,ZD,
        EDIT=(SIIT.T),SIGNS=(+,-))
  ```

- INREC, OUTREC and OUTFIL can generate sequence numbers in output records. You can create BI, PD, ZD or CSF/FS sequence numbers. Starting values and increment values can be defaulted or specified. You can restart the sequence number at the starting value whenever a value in a specified field changes. As a simple example, if you specify:

  ```
    INREC OVERLAY=(81:SEQNUM,8,ZD,START=1000,INCR=100,RESTART=(25,5))
  ```

  zoned decimal sequence numbers 00001000, 00001100, 00001200, and so on will be generated in positions 81-88 of the output records. Each time the value in positions 25-29 changes, the sequence number will start again at 00001000. Note that an F-sign is used for zoned decimal sequence numbers so they will be displayable and printable.

- By default, DFSORT uses the OUTREC or INREC record length as the SORTOUT LRECL when the SORTOUT LRECL is unavailable, instead of using the SORTIN length. This can eliminate unexpected/unwanted padding and truncation. For example, previous to this change, if you had a fixed-length input data set with an LRECL of 80 and specified:

  ```
     OUTREC BUILD=(1,50)
  ```

  you'd get a SORTOUT LRECL of 80 and output records padded with binary zeros from positions 51-80. Now, instead by default, you'll get a SORTOUT LRECL of 50 and no padding.

  The **SOLRF** installation and run-time options allow you to specify whether you want the SORTOUT LRECL to be set the new way or old way when INREC or OUTREC is specified, as follows:

  – **SOLRF=YES** (installation) or **SOLRF** (run-time) tells DFSORT to use the INREC or OUTREC length as the SORTOUT LRECL. This is the IBM-supplied default and is usually what you want when you specify INREC or OUTREC. This is also the way OUTFIL OUTREC always works.

  – **SOLRF=NO** (installation) or **NOSOLRF** (run-time) tells DFSORT not to use the INREC or OUTREC length as the SORTOUT LRECL. This is the way DFSORT used to work and may cause unexpected/unwanted padding or truncation of the SORTOUT records.

OUTFIL-Only Features

In addition to the reformatting features mentioned earlier, many other new features are available with DFSORT's versatile OUTFIL multiple output and reporting function, giving it added flexibility.

- The **IFTRAIL, TRLID, TRLUPD and HD=YES** parameters of OUTFIL let you update count and total values in an existing trailer (last) record when you add, delete or modify data records.

  Here's an example of how you could use IFTRAIL to split an input file with a header, and a trailer containing a count and total for the input data records, into two output files each of which has the header and a trailer with accurate count and total for the included data records:

  ```
  OUTFIL FNAMES=OUT1,
   INCLUDE=(3,4,CH,EQ,C'key1'),
   IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'T'),
    TRLUPD=(6:COUNT=(M11,LENGTH=4),
           14:TOT=(10,4,ZD,TO=ZD,LENGTH=6)))
  OUTFIL FNAMES=OUT2,
   INCLUDE=(3,4,CH,EQ,C'key2'),
   IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'T'),
    TRLUPD=(6:COUNT=(M11,LENGTH=4),
           14:TOT=(10,4,ZD,TO=ZD,LENGTH=6)))
  ```

  OUT1 will contain the input header record, the input data records with 'key1' in positions 3-6, and the input trailer record with updated count and total values for the included 'key1' records.

  OUT2 will contain the input header record, the input data records with 'key2' in positions 3-6, and the input trailer record with updated count and total values for the included 'key2' records.

- The **ACCEPT=n** parameter of OUTFIL lets you limit the number of records accepted for OUTFIL processing. A record is "accepted" if it is not deleted by STARTREC, ENDREC, SAMPLE, INCLUDE or OMIT processing.

  Here's an example of ACCEPT processing:

  ```
   OUTFIL FNAMES=OUT1,INCLUDE=(11,3,CH,EQ,C'D51')
   OUTFIL FNAMES=OUT2,STARTREC=3,ACCEPT=5
  ```

  The first 5 'D51' records are written to the OUT1 data set. Relative records 3 through 7 are written to the OUT2 data set.

- The **SPLIT1R=n, SPLITBY=n** and **SPLIT** parameters of OUTFIL let you distribute records among your OUTFIL data sets in various ways.

  SPLIT1R=n writes n records to each OUTFIL data set in turn. Extra records are written to the last OUTFIL data set, so each OUTFIL data set contains contiguous records.

  Here's an example of splitting records with SPLIT1R:

  ```
  OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLIT1R=50
  ```

  If the input data set contains 158 records, records 1-50 are written to OUT1, records 51-100 are written to OUT2 and records 101-158 are written to OUT3.

  SPLIT writes one record to each OUTFIL data set in turn and then starts over again with the first OUTFIL data set. SPLITBY=n writes n records to each OUTFIL data set in turn and then starts over again with the first OUTFIL data set. Thus, each OUTFIL data set may not contain contiguous records. SPLIT and SPLITBY=1 are equivalent.

  Here's an example of splitting records with SPLIT and SPLITBY=n:

```
OUTFIL FNAMES=(PIPE1,PIPE2,PIPE3),SPLIT
OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLITBY=50
```

The first record is written to the PIPE1 pipe, the second record is written to the PIPE2 pipe, the third record is written to the PIPE3 pipe, the fourth record is written to the PIPE1 pipe, and so on.

Records 1-50 are written to the OUT1 data set, records 51-100 are written to the OUT2 data set, records 101-150 are written to the OUT3 data set, records 151-200 are written to the OUT1 data set, and so on.

- Signed hexadecimal floating point values can be converted into their corresponding signed integer values for calculating total, minimum, maximum and average statistics with OUTFIL's TRAILERx parameters. The integer values can be further edited using edit masks or edit patterns, or further converted to BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or FS/CSF values. This makes it easy to display statistics for hexadecimal floating point values.

  Here's an example of displaying a total and average for a hexadecimal floating point value:

  ```
   OUTFIL TRAILER1=(C'Total:',10:TOT=(51,8,FL,M12),/,
                   C'Average:',10:AVG=(51,8,FL,M12))
  ```

- The **BLKCCH1, BLKCCH2** and **BLKCCT1** parameters of OUTFIL can be used to suppress page ejects for reports in various ways.

  BLKCCH1 allows you to avoid forcing a page eject at the start of the report header; the ANSI carriage control character of '1' (page eject) in the first line of the report header (HEADER1) is replaced with a blank.

  BLKCCH2 allows you to avoid forcing a page eject at the start of the first page header; the ANSI carriage control character of '1' (page eject) in the first line of the first page header (HEADER2) is replaced with a blank.

  BLKCCT1 allows you to avoid forcing a page eject at the start of the report trailer; the ANSI carriage control character of '1' (page eject) in the first line of the report trailer (TRAILER1) is replaced with a blank.

  For example, with:

  ```
  OUTFIL BLKCCH2,
    HEADER1=('Control report',2X,DATE,2/),
    HEADER2=('Page ',PAGE=(EDIT=(IIT)),/,
      'Account',12:'Revenue')
  ```

  DFSORT does a page eject at the start of the report header, and at the start of the second and subsequent page headers, but suppresses the page eject at the start of the first page header. Thus, the report header and first page header appear on the same page.

- OUTFIL can produce reports without the ANSI carriage control characters that indicate actions to be taken on a printer (for example, page eject, skip a line, and so on). If the **REMOVECC** operand is specified, DFSORT "removes" the carriage control character from each record of the report. This makes it easy to remove the printer controls when they're not needed because the output will be viewed or written to a list data set, rather than printed. When REMOVECC is specified, the RECFM does not need to include 'A' for ANSI (for example, it can be FB instead of FBA) and the LRECL does not need to include an extra byte for the carriage control character.

  As a simple example, if you specify the following for an input data set with 5723 records:

  ```
  OUTFIL FNAMES=TOTCNT,NODETAIL,
    TRAILER1=(COUNT=(M11,LENGTH=6))
  ```

  TOTCNT would contain:

  ```
  1005723
  ```

  where '1' is the carriage control character for a page eject and '005723' is the count of total records. If you were passing the count to another program that expected it to start in column 1, you could specify:

```
  OUTFIL FNAMES=TOTCNT,NODETAIL,REMOVECC,
    TRAILER1=(COUNT=(M11,LENGTH=6))
```

to remove the carriage control character so TOTCNT would contain:

```
005723
```

- OUTFIL can add or subtract any number from 1 to 999 to the counts in TRAILERx records. **COUNT+n=(edit)** adds n to the count and edits it using the specified edit mask or edit pattern. **COUNT+n=(to)** adds n to the count and converts it to the specified BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or CSF/FS value. **COUNT-n=(edit)** subtracts n from the count and edits it using the specified edit mask or edit pattern. **COUNT-n=(to)** subtracts n from the count and converts it to the specified BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or CSF/FS value. For example, if the input data set has a header record, data records and a trailer record, you can use:

  ```
    OUTFIL TRAILER1=('Count of data records is',
      COUNT-2=(TO=FS,LENGTH=7)
  ```

  to get the count of the data records (rather than the count of all of the records) as a 7-byte FS value.

- The numbers generated for **COUNT, TOT, MAX, MIN, AVG, SUBCOUNT, SUBTOT, SUBMIN, SUBMAX** and **SUBAVG** in OUTFIL TRAILERx records, and for **PAGE** in OUTFIL HEADERx and TRAILERx records, can be edited with edit masks or edit patterns, or converted to BI, FI, PD, PDF, PDC, ZD, ZDF, ZDC or CSF/FS values. In addition, **hexadecimal strings** (X'yy...yy' or nX'yy...yy') can be inserted in HEADERx and TRAILERx records. For example, with:

  ```
   OUTFIL FNAMES=OUT,
    TRAILER1=(TOT=(EDIT=(III,IIT.TT)),5X'FF',COUNT=(TO=PD,LENGTH=5))
  ```

  a total of 123456 appears as ' 1,234.56' in output positions 1-10, X'FFFFFFFFFF' appears in positions 11-16 and a count of 5032 is is converted to a 5-byte value of P'5032' (X'00005032C') in output positions 17-21.

- OUTFIL can generate current date constants in various forms in HEADERx and TRAILERx records, with or without separators, using the parameters **DATE, DATE=(abcd), DATENS=(abc), YDDD=(abc)** and **YDDDNS=(ab)**, where a, b, c and d represent various parts of the date (mm, dd, yy, yyyy, ddd) and various separators (for example, ., - or /). OUTFIL can generate current time constants in various forms in HEADERx and TRAILERx records, with or without separators, using **TIME, TIME=(abc)** and **TIMENS=(ab)** where ab represents 12-hour or 24-hour time and c represents various separators (for example, :, . or -).

- OUTFIL OUTREC or BUILD can create many output records from each input record in any OUTFIL data set. You can split the input record into pieces (for example, put the first 20 bytes into the first output record and the last 60 bytes into the second output record), use the input fields in one or more of the output records, double or triple space in reports, and so on.

  /, /.../ and **n/** separators in the OUTFIL OUTREC or BUILD operand allow you to start a new record or insert blank records. As a simple example, if you specify:

  ```
     OUTFIL BUILD=(2/,C'Field 2 contains ',4,3,/,
          C'Field 1 contains ',1,3)
  ```

  an input data set containing these records:

  ```
  AAABBB
  CCCDDD
  ```

  would produce an output data set containing these records:

```
blanks
blanks
Field 2 contains BBB
Field 1 contains AAA
blanks
blanks
Field 2 contains DDD
Field 1 contains CCC
```

Note that **four** output records are produced for each input record.

- The **SAMPLE=n** and **SAMPLE=(n,m)** parameters of OUTFIL can be used to sample records in a variety of ways. The sample consists of the first m records in every nth interval. STARTREC=x and ENDREC=y can be used with SAMPLE to select a range of records to be sampled. SAMPLE=n writes every nth record starting at the STARTREC record and ending at or before the ENDREC record. SAMPLE=(n,m) writes m records every nth record starting at the STARTREC record and ending at or before the ENDREC record.

  Here's an example of OUTFIL sampling:

  ```
    OUTFIL FNAMES=OUT1,SAMPLE=5
    OUTFIL FNAMES=OUT2,SAMPLE=(1000,2),ENDREC=2500
    OUTFIL FNAMES=OUT3,STARTREC=23,ENDREC=75,SAMPLE=25
    OUTFIL FNAMES=OUT4,STARTREC=1001,SAMPLE=(100,3)
  ```

  The input records written to each output data set are as follows (remember that the default for STARTREC is 1 and the default for ENDREC is the last record in the data set):

  – OUT1: 1, 6, 11, and so on

  – OUT2: 1, 2, 1001, 1002, 2001, 2002

  – OUT3: 23, 48, 73

  – OUT4: 1001, 1002, 1003, 1101, 1102, 1103, and so on

- The **REPEAT=n** parameter of OUTFIL lets you write each output record multiple times. The repeated records are identical, unless the SEQNUM operand is used to create different sequence numbers for the repeated records.

  Here's an example of OUTFIL repetition:

  ```
    OUTFIL FNAMES=RPT50,REPEAT=5000
    OUTFIL FNAMES=RPTSQ,REPEAT=20,OVERLAY=(81:SEQNUM,8,ZD)
  ```

  5000 identical copies of each input record are written to the RPT50 output data set.

  20 reformatted output records are written to the RPTSQ output data set for each input record. The reformatted output records consist of input positions 1-80 and an 8-byte ZD sequence number that starts at 1 and is incremented by 1 for each output record, including the repeated records.

- OUTFIL can convert fixed-length input records (for example, FB) to variable-length output records (for example, VB). If the new **FTOV** operand is specified without **OUTREC** or **BUILD**, the entire fixed-length record is used to build the variable-length record. If **FTOV** is used with **OUTREC** or **BUILD**, the specified fields from the fixed-length record are used to build the variable-length record. This makes it easy to use all of DFSORT's features when converting from FB to VB.

  For example, FTOV might be used as follows:

  ```
    OUTFIL FNAMES=VAROUT,FTOV
    OUTFIL FNAMES=V1,FTOV,BUILD=(1,20,26:21,10,6C'*')
  ```

  For ease-of-use, **VTOF** can be used as an alias for **CONVERT**. Thus, FTOV can be used to convert from FB to VB, and VTOF (or CONVERT) can be used to convert from VB to FB.

- OUTFIL can remove trailing bytes such as spaces, binary zeros, or asterisks from variable-length records. The new **VLTRIM=byte** operand tells DFSORT to remove trailing bytes of the specified type from variable-length output records. The trim byte can be any value. If DFSORT finds one or more trim bytes at the end of a variable-length output record, it removes those bytes by decreasing the length of the record. This makes it easy to get rid of unwanted padding bytes at the end of VB records.

  For example, you could use:

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

  to convert these 17-byte FB records:

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

  to these VB records (4-byte RDW followed by data):

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

- OUTFIL can process short variable-length input records. Any missing bytes you specify for the output record can be filled with a padding byte you specify.

  The new **VLFILL=byte** operand allows DFSORT to continue processing if a variable-length record is too short to contain all specified OUTREC or BUILD fields. The fill byte can be any value. Missing bytes in OUTREC or BUILD fields are replaced with the specified fill byte so the filled fields can be processed. For example, VLFILL=byte might be used as follows:

  ```
     OUTFIL FNAMES=VB1,VLFILL=X'FF',BUILD=(1,4,15,5,52)
     OUTFIL FNAMES=FB1,CONVERT,BUILD=(1,20,2X,35,10),VLFILL=C'*'
  ```

  The VB1 output data set will be reformatted. Missing bytes in positions 15-19 will be replaced with FF bytes in the reformatted variable-length output records.

  The FB1 output data set will be reformatted and converted from variable-length to fixed-length. Missing bytes in positions 1-20 or 35-44 of the variable-length input records will be replaced with asterisks in the reformatted fixed-length output records.

  When you use CONVERT or VTOF without specifying VLFILL=byte, VLFILL=C' ' is automatically used as the default. For example, if you specify:

  ```
     OUTFIL FNAMES=CNVB,CONVERT,BUILD=(1,65)
  ```

  the CNVB output data set will be reformatted and converted from variable-length to fixed-length. Missing bytes in positions 1-65 of the variable-length input records will be replaced with spaces in the reformatted fixed-length output records.

INCLUDE and OMIT Features

DFSORT's INCLUDE and OMIT statements, and the OUTFIL INCLUDE and OMIT operands, are more powerful and flexible than ever.

- INCLUDE and OMIT can test fields for numerics (field,EQ,NUM) or non-numerics (field,NE,NUM) of the following forms:

- Character (FS format): Test for '0'-'9' in all bytes. For example, if you used the following INCLUDE statement:

  ```
  INCLUDE COND=(21,5,FS,EQ,NUM)
  ```

  a record with a value of '02579' in positions 21-25 would be included, whereas records with values of '02A79', '-2579' and '0257 ' would not be included.

- Zoned decimal (ZD format): Test for X'F0'-X'F9' for all bytes except the last, and X'F0'-X'F9', X'D0'-X'D9' or X'C0'-X'C9' in the last (sign) byte. For example, if you used the following OMIT statement:

  ```
  OMIT COND=(31,3,ZD,NE,NUM)
  ```

  a record with a value of X'F1F3D8' in positions 31-33 would not be omitted, whereas records with values of X'F1F3A8' and X'404040' would be omitted.

- Packed decimal (PD format): Test for 0-9 for all digits and F, D or C for the sign. For example, if you used the following INCLUDE statement:

  ```
  INCLUDE COND=(11,2,PD,EQ,NUM)
  ```

  a record with a value of X'832C' in positions 11-12 would be included, whereas records with values of X'3A2F' and X'4290' would not be included.

This makes it easy to include or omit records based on whether they contain valid numeric data.

- INCLUDE and OMIT can compare appropriate fields against generated run-time constants for current, past and future dates in the following forms (d is days, m is months and c is any character except a blank):

Operand	**Constant**
DATE1, DATE1-d, DATE1+d	C'yyyymmdd'
DATE1(c), DATE1(c)-d, DATE1(c)+d	C'yyyycmmcdd'
DATE1P, DATE1P-d, DATE1P+d	+yyyymmdd
DATE2, DATE2-m, DATE2+m	C'yyyymm'
DATE2(c), DATE2(c)-m, DATE2(c)+m	C'yyyycmm'
DATE2P, DATE2P-m, DATE2P+m	+yyyymm
DATE3, DATE3-d, DATE3+d	C'yyyyddd'
DATE3(c), DATE3(c)-d, DATE3(c)+d	C'yyyycddd'
DATE3P, DATE3P-d, DATE3P+d	+yyyyddd
DATE4	C'yyyy-mm-dd-hh.mm.ss'
Y'DATE1', Y'DATE1'-d, Y'DATE1'+d	Y'yymmdd'
Y'DATE2', Y'DATE2'-m, Y'DATE2'+m	Y'yymm'
Y'DATE3', Y'DATE3'-d, Y'DATE3'+d	Y'yyddd'

This makes it easy to include or omit records based on whether they contain dates equal to, lower than or higher than the date of the run, a date before the run or a date after the run.

DATEn and DATEn(c) generate a character string (C'string') for today's date that can be used in comparisons just like any other character string. Likewise, DATEn-r and DATEn(c)-r generate a character string for a past date and DATEn+r and DATEn(c)+r generate a character string for a future date.

DATEnP generates a decimal number (+n) for today's date that can be used in comparisons just like any other decimal number. Likewise, DATEnP-r generates a decimal number for a past date and DATEnP+r generates a decimal number for a future date.

Y'DATEn' generates a Y constant (Y'string') for today's date that can be used in date comparisons just like any other Y constant. Likewise, Y'DATEn'-r generates a Y constant for a past date and Y'DATEn'+r generates a Y constant for a future date.

For example, if you used the following INCLUDE statement for a DFSORT run on January 29, 2006:

```
INCLUDE COND=(21,8,ZD,GE,DATE1P-7,AND,21,8,ZD,LE,DATE1P)
```

the generated date for DATE1P-7 would be +20060122 and the generated date for DATE1P would be +20060129. The SORTOUT data set would include only those records with a date in positions 21-28 between +20060122 and +2006129. So a record with a Z'20060125' date would be included, whereas a record with a Z'20060120' date would not be included.

- The maximum length of an SS (substring compare) field has been raised from 256 bytes to 32752 bytes. This makes it easy to check for a constant anywhere in a large field or in an entire record.

  Here's an example of substring compare for an entire 10000-byte record:

  ```
  INCLUDE FORMAT=SS,
    COND=(1,10000,EQ,C'Error',OR,
          1,10000,EQ,C'Warning')
  ```

- INCLUDE and OMIT can compare a binary (BI) field to a decimal constant (n or +n) as well as to a character or hexadecimal string. This makes it much easier to specify INCLUDE and OMIT conditions for binary numbers. For example, you can use this simple statement with a decimal constant to only include VB records with a length less than 220 bytes:

  ```
  INCLUDE COND=(1,2,BI,LT,220)
  ```

  instead of this more complicated statement using a hexadecimal constant:

  ```
  INCLUDE COND=(1,2,BI,LT,X'00DC')
  ```

- The maximum position for the end of an INCLUDE or OMIT field has been raised from 4092 to 32752. This makes it easy to use INCLUDE and OMIT conditions for fields almost anywhere in your records.

- The number of conditions you can use with INCLUDE and OMIT has been increased significantly. You can use more conditions for field-to-field, field-to-constant, substring and bit logic tests. This allows you to increase the complexity of the criteria you use to determine which records will be kept or deleted for a sort, copy or merge application.

- You have more choices for handling "short" INCLUDE/OMIT compare fields. A short field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record.

  The new installation option **VLSCMP=YES/NO** and run-time options **VLSCMP/NOVLSCMP** together with the previously available installation option **VLSHRT=YES/NO** and run-time options **VLSHRT/NOVLSHRT** provide three levels of processing for short INCLUDE/OMIT fields in the following hierarchy:

  1. If VLSCMP is in effect, DFSORT pads short INCLUDE/OMIT fields with binary zeros temporarily for comparison testing. This allows all of the INCLUDE/OMIT comparisons in a logical expression to be performed, even if some fields are short. Since short fields are padded with binary zeros, comparisons involving short fields are false. Comparisons involving non-short fields can be true or false.

     Note: In cases where padding of short fields with binary zeros may result in unwanted true comparisons, you can get the result you want by adding an appropriate check of the record length to the INCLUDE/OMIT logical expression.

2. If NOVLSCMP and VLSHRT are in effect, DFSORT treats the entire INCLUDE/OMIT logical expression as false if any field is short. Thus, comparisons involving non-short fields are ignored if any comparison involves a short field.
3. If NOVLSCMP and NOVLSHRT are in effect, DFSORT issues an error message, terminates and gives a return code of 16 if a short INCLUDE/OMIT field is found. VLSCMP=NO and VLSHRT=NO are the IBM-supplied installation defaults.

To illustrate how this works, suppose you specify:

```
INCLUDE COND=(6,1,CH,EQ,C'1',OR,70,2,CH,EQ,C'T1')
```

If a variable-length input record has a length less than 71 bytes, the field at bytes 70-71 is short.

– If you specify:

  ```
  OPTION VLSCMP
  ```

  the record is included if byte 6 of the input record is C'1' or omitted if byte 6 is not C'1'. The comparison of bytes 70-71 equal to C'T1' is false because bytes 70-71 contain either X'hh00' (for a record length of 70 bytes) or X'0000' (for a record length of less than 70 bytes). The comparison involving the non-short field is performed even though a short field is present.

– If you specify:

  ```
  OPTION NOVLSCMP,VLSHRT
  ```

  the record is omitted because any short field makes the entire logical expression false. The comparison involving the non-short field is not performed because a short field is present.

– If you specify:

  ```
  OPTION NOVLSCMP,NOVLSHRT
  ```

  DFSORT terminates because any short field results in termination.

Symbols for Fields and Constants

DFSORT gives your site a powerful, simple and flexible way to use symbols in DFSORT and ICETOOL statements. Now you can create and use symbol mappings for your own frequently used data.

DFSORT symbols turn DFSORT's syntax into a high level language. Symbols can help to standardize your DFSORT applications and increase your productivity. You can use a symbol anywhere you can use a field or constant in any DFSORT control statement or ICETOOL operator. DFSORT symbols can be up to 50 characters, are case-sensitive and can include underscore (_) and hyphen (-) characters. Thus, you can create meaningful, descriptive names for your symbols, such as Price_of_Item (or Price-of-Item) making them easy to remember, use and understand.

Field symbols define a field in terms of its position, length and format. Constant symbols define a field in terms of its literal, numeric or bit value. Once you make a symbol available, you free yourself from the sometimes tedious process of figuring out its position, length, format or value. No more confusion over offsets versus positions and whether to add 4 for the RDW or not. No more recoding positions in statements when you rearrange fields in your data.

To give you a quick idea of how easy it is to use DFSORT symbols, let's look at the JCL and control statements for a simple DFSORT job tha uses symbols for fields and constants.

Here's a DFSORT symbols data set named ACCOUNT.SYMBOLS you might have created to map the fields and constants for your ACCOUNT data set (note that the statements shown here only cover a few of the many features of DFSORT's easy to use and flexible symbol mapping syntax).

```
ACCOUNT.SYMBOLS data set:

* Fields for ACCOUNT data set records.
*   '*' for position means use next location.
*   '=' for position, length or format means use corresponding value
*       from the previous field.
Full_Name,1,40,CH
  First_Name,=,20,CH    Subfield of Full_Name
  Last_Name,*,=,=       Subfield of Full_Name
Account_Number,*,3,PD
SKIP,2                  Skip 2 unused bytes
Balance,*,6,ZD
  Level1,50000          Decimal constant for Balance
  Penalty,-100          Decimal constant for Balance
Type,*,8,CH
  Loan,'LOAN'           Character constant for Type
  Check,'CHECKING'      Character constant for Type
```

Here's the JCL and control statements for a DFSORT job that uses the symbols in ACCOUNT.SYMBOLS:

```
//EXAMP JOB ...
//RUNIT EXEC PGM=ICEMAN
//** SYMNAMES POINTS TO ONE OR MORE SYMBOL DATA SETS.
//SYMNAMES DD DSN=ACCOUNT.SYMBOLS,DISP=SHR
//** SYMNOUT LISTS THE ORIGINAL SYMBOL STATEMENTS AND
//** THE SYMBOL TABLE DFSORT BUILDS FROM THEM.
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=ACCOUNT,DISP=SHR
//SORTOUT DD ...
//SYSIN DD *
  INCLUDE COND=((Type,EQ,Loan,AND,Balance,GT,Level1),OR,
                (Type,EQ,Check,AND,Balance,LE,Penalty))
  SORT FIELDS=(Full_Name,A,Type,A,Account_Number,D)
/*
```

Symbols for Output Columns

INREC, OUTREC and OUTFIL allow you to use symbols for output columns. You can use symbol: for an output column wherever you can use c: for an output column (c is the output position). The symbol can define a field in terms of position (p) or position and length (p,m) or position, length and format (p,m,f).

The following example shows how symbols can be used for output columns.

SYMNAMES:

```
Amt,8,5,ZD
Dept,27,5
New_col,52
```

Control statements:

```
  OPTION COPY
  INREC OVERLAY=(New_col:Amt,MUL,+2,TO=FS,LENGTH=8,
     Dept:Dept,JFY=(SHIFT=RIGHT),
     Amt:Amt,TO=FS,LENGTH=8)
```

Symbols for %nn Parsed Fields

INREC, OUTREC and OUTFIL allow you to use symbols for %nn parsed fields. You can define a symbol for a %nn field (%00-%99) and use that symbol anywhere you can use %nn. The following example shows how symbols can be used for %nn fields.

SYMNAMES:

```
Name,%00
Amt,%01
Dept,%02
Name_col,5
Amt_col,21
Dept_col,41
```

Control statements:

```
  OPTION COPY
  OUTREC PARSE=(Name=(ENDBEFR=C',',FIXLEN=10),
                %=(ENDBEFR=C','),
                Amt=(ENDBEFR=C',',FIXLEN=8),
                Dept=(ENDBEFR=C',',FIXLEN=12)),
         BUILD=(Name_col:Name,
                Amt_col:Amt,SFF,TO=FS,LENGTH=10,
                Dept_col:Dept,TRAN=UTOL)
```

System Symbols

You can use system symbols in character constants defined as symbols. symbol,s'string' or symbol,S'string' can be used to define a string containing any combination of EBCDIC characters and system symbols you want to use to form a character string. You can use dynamic system symbols (e.g. &JOBNAME and &DAY), system-defined static symbols (e.g. &SYSNAME and &SYSPLEX) and static system symbols defined by your installation.

The following example shows how system symbols can be used in symbol constants:

SYMNAMES:

```
Title,S'Jobname: &JOBNAME., Sysplex: &SYSPLEX., System: &SYSNAME.,'
Totday,S'Total for &WDAY.:'
```

Control statements:

```
  OPTION COPY
  OUTFIL REMOVECC,
   HEADER2=(Title,X,' Page: ',PAGE=(EDIT=(TT)),/),
   TRAILER1=(/,Totday,20:TOT=(21,4,FS,TO=FS,LENGTH=5))
```

SET and PROC Symbols

You can construct DFSORT symbols (JP0-JP9) that incorporate JCL PROC or SET symbols as well as text and system symbols, and use those constructed symbols in DFSORT and ICETOOL control statements.

The following example shows how SET symbols XDSN and YDSN can be used in a DFSORT OMIT statement:

```
// SET XDSN='X.DATA',YDSN='Y.DATA'
//S1 EXEC PGM=SORT,PARM='MSGDDN=MYOUT,JP1"&XDSN",JP2"&YDSN",LIST'
//MYOUT DD SYSOUT=*
...
//SYSIN DD *
  OPTION COPY
  OMIT COND=(1,44,CH,EQ,JP1,OR,1,44,CH,EQ,JP2)
/*
```

Symbols for RACF, DFSMSrmm and DCOLLECT

To increase your productivity, IBM's DFSORT, RACF and DFSMS teams have already created DFSORT symbol mappings and sample jobs for data associated with RACF, DFSMSrmm and DCOLLECT. These mappings and jobs further enhance ICETOOL's usefulness as the analysis and reporting tool of choice for data associated with these products:

- RACF

  RACF's RACFICE2 package contains the tools necessary to create reports using the output of IRRDBU00 and IRRADU00 as input to DFSORT's ICETOOL utility. RACFICE2 includes DFSORT Symbol mappings for IRRDBU00 and IRRADU00. You can download this package from the RACFICE2 home page at:

  www.ibm.com/systems/z/os/zos/features/racf/downloads/racfice.html

- DFSMSrmm

  DFSMSrmm provides you with symbols you can use in DFSORT and ICETOOL jobs to create reports for DFSMSrmm-managed resources. These symbol mappings are available in SYS1.MACLIB after SMP/E APPLY processing, as members EDGACTSY, EDGEXTSY, and EDGSMFSY.

- DCOLLECT

  You can download the DFSORT Symbols for DCOLLECT from:

  ftp://ftp.software.ibm.com/storage/dfsort/mvs/symbols/

SUM Features and Extensions

Important new features are available for DFSORT's SUM statement.

- The maximum position for the end of a SUM field has been raised from 4092 to 32752. This makes it easy to use SUM for fields almost anywhere in your records.
- The usefulness of **VLSHRT** has been extended by allowing it to be used to handle "short" SUM fields. A "short" field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record. When VLSHRT is in effect, DFSORT will leave short SUM fields unsummed. In addition short SORT, MERGE, INCLUDE and OMIT fields can be processed with VLSHRT even when a SUM statement is present.

- The number of fields you can use with the frequently used SUM statement has been increased significantly. This allows you to increase the number of fields you total for a sort or merge application.
- You have new ways to handle an overflow condition for SUM fields. The new **OVFLO** installation and run-time option allows you to specify what you want DFSORT to do when BI, FI, PD or ZD sum fields overflow, as follows:
  - **OVFLO=RC0** tells DFSORT to issue an informational message, set a return code of **0** and continue processing when sum fields overflow. The pair of records involved in the overflow is left unsummed.
  - **OVFLO=RC4** tells DFSORT to issue an informational message, set a return code of **4** and continue processing when sum fields overflow. The pair of records involved in the overflow is left unsummed.
  - **OVFLO=RC16** tells DFSORT to issue an error message, terminate and give a return code of **16** if sum fields overflow.

SORT and MERGE Extensions

DFSORT's SORT and MERGE control fields are more flexible.

- The maximum position for the end of a SORT or MERGE field has been raised from 4092 to 32752. This makes it easy to use SORT and MERGE for fields almost anywhere in your records.
- The length for SORT and MERGE fields with ZD (zoned decimal) and PD (packed decimal) format has been raised from 32 bytes to 256 bytes. This makes it easier to sort or merge using longer ZD and PD fields.
- The length for SORT and MERGE fields with AQ (alternate collating sequence) and AC (ISCII/ASCII character) format has been raised from 256 bytes to 4092 bytes. This makes it easier to sort or merge using longer AQ and AC fields.

Larger Fields and Constants

DFSORT and ICETOOL can handle up to 31-digit signed values for ZD, PD and FS fields and decimal constants, up to 8-byte signed values for FI fields, and up to 8-byte unsigned values for BI fields, for all cases. This is especially useful for doing various types of processing for large COBOL values like PIC S9(20)V9(5) and PIC S9(18) COMP-3.

For example, if you wanted to display the maximum, minimum, average and total of a 25-byte ZD field and an 18-byte PD field in your input records, you could use this ICETOOL operator:

```
  STATS FROM(IN) ON(5,25,ZD) ON(41,18,PD)
```

Free Form Formats

Two important formats are available for DFSORT and ICETOOL that allow you to extract the sign and digits from a wide variety of free form fields, ignoring any other characters in those fields. These new formats make it easy to do various types of processing for numeric values from up to 44-character fields containing decimal points, separators, leading or trailing signs, currency signs, and so on.

- The **SFF** (signed free form numeric) format extracts decimal digits (0-9) from right to left anywhere in the field to form a positive or negative number. If - or) is found anywhere in the field, the number is treated as negative, otherwise it is treated as positive. Any combination of characters is valid, but characters other than 0-9, - and) are ignored.

  For example, if you had the following values in your input records:

```
 $358,253.052
  ($1,860.108)
      ($5.072)
  $28,075.936
```

you could sort the values correctly using the following SORT statement:

```
  SORT FIELDS=(1,14,SFF,A)
```

- The **UFF** (unsigned free form numeric) format extracts decimal digits (0-9) from right to left anywhere in the field to form a positive number. Any combination of characters is valid, but characters other than 0-9 are ignored.

  For example, if you had the following values in your input records:

  ```
  2005/03/13
  2005.02.28
  2004-12-18
  2005/02/16
  ```

  you could keep the records with a date higher than or equal to 2005/02/28 using the following INCLUDE statement:

  ```
    INCLUDE COND=(1,10,UFF,GE,+20050228)
  ```

Managed Tape Processing

DFSORT significantly improves the way tapes are processed when DFSORT can obtain information about tape data sets from a tape management system. DFSORT can obtain such information automatically from **DFSMSrmm**, but an **ICETPEX routine** is required to obtain the same information from other tape management systems. Check with your tape management vendor to find out if they currently have an ICETPEX routine available or have plans to provide one in the future.

DFSORT can use the information passed to it from DFSMSrmm or ICETPEX, when appropriate, to improve its processing of managed tapes in the following ways:

- DFSORT can obtain accurate input filesize information for managed tapes. This can result in improved sort performance and more accurate dynamic workspace allocation.

  Additionally, you don't have to supply the input filesize to DFSORT when this information is available from DFSMSrmm or ICETPEX. DFSORT will automatically use the filesize it obtains from DFSMSrmm or ICETPEX to override any **FILSZ=En** or **SIZE=En** value you specify. However, you must remove any **FILSZ=n, FILSZ=Un, SIZE=n** or **SIZE=Un** value you specify in order for DFSORT to use the filesize it obtains from DFSMSrmm or ICETPEX.

- DFSORT can obtain input and output attributes such as RECFM, LRECL and BLKSIZE for managed tapes. As a result, you don't have to specify these attributes explicitly for input and output tape data sets when this information is available from DFSMSrmm or ICETPEX.

Improvements for RACF's IRRUT200 Utility

The DFSORT copy function can be used when **ICEGENER** is called by a program that uses an alternate SYSIN ddname with DUMMY. This can result in improved performance for **RACF's IRRUT200 utility** when ICEGENER is installed as a replacement for IEBGENER.

64-Bit Architecture

DFSORT can exploit 64-bit real architecture by using **memory objects** for sort applications, when appropriate.

DFSORT can exploit 64-bit real architecture by backing storage and data spaces in real storage above 2 gigabytes, and by using central storage instead of expanded storage for Hipersorting.

Multiple Hiperspaces

DFSORT can use multiple Hiperspaces for external storage requirements, increasing DFSORT's ability to use Hipersorting for large sort applications when sufficient system resources are available.

Easier Migration from Other Sort Products

DFSORT provides options and features that make it easier to migrate to DFSORT from other sort products. Many of the new and previously available migration features incorporated into DFSORT make it operate like other sort products automatically. However, the options shown in Table 1 have IBM-supplied installation defaults, as indicated, that you may want to change to make DFSORT operate more like the sort product you are migrating from.

Changing an installation option (via an ICEPRMxx member in PARMLIB) changes the way DFSORT works globally by default. Specifying a run-time option changes the way DFSORT works for a specific application.

Table 1 (Page 1 of 2). Options That Can Make Migration Easier

Installation Option	**Run-Time Option**	**Specifies ...**

ABCODE=MSG
ABCODE=n
Default: MSG
		the ABEND code for a critical error.

DYNALOC=(d,n) Default: SYSDA,4	DYNALLOC=(d,n)	the device name and maximum number of dynamically allocated work data sets.

DYNAUTO=YES
DYNAUTO=IGNWKDD
DYNAUTO=NO
Default: YES
	DYNALLOC=(d,n)	whether work data sets are dynamically allocated.

DYNSPC=n Default: 256	DYNSPC=n	the dynamically allocated work space when the file size is unkn.

EQUALS=YES
EQUALS=NO
EQUALS=VBLKSET
Default: VLBLKSET
	EQUALS
	NOEQUALS
		whether the order of records that collate identically is preserved from input to output.

EXITCK=STRONG
EXITCK=WEAK
Default: STRONG
	EXITCK=STRONG
	EXITCK=WEAK
		whether DFSORT terminates or continues for invalid return codes from E15/E35 user exits.

FSZEST=YES
FSZEST=NO
Default: NO
	FILSZ=n
	FILSZ=En
	FILSZ=Un
		whether DFSORT treats file sizes as exact or estimated.

Table 1 (Page 2 of 2). Options That Can Make Migration Easier

Installation Option	**Run-Time Option**	**Specifies ...**

NOMSGDD=QUIT
NOMSGDD=ALL
NOMSGDD=CRITICAL
NOMSGDD=NONE
Default: QUIT
		whether DFSORT terminates or continues when the message data set is not available.

PARMDDN=ddname Default: DFSPARM		an alternate ddname, such as $ORTPARM, for the DFSPARM control data set.

RESET=YES
RESET=NO
Default: YES
	RESET
	NORESET
		whether DFSORT processes a VSAM set defined with REUSE as NEW or MOD.

SORTLIB=SYSTEM
SORTLIB=PRIVATE
Default: PRIVATE
		whether DFSORT searches a system or private library for tape work data set sort or Conventional merge modules.

SZERO=YES
SZERO=NO
Default: YES
	SZERO
	NOSZERO
		whether DFSORT treats zero values as signed or unsigned.

VLLONG=YES
VLLONG=NO
Default: NO
	VLLONG
	NOVLLONG
		whether DFSORT truncates long variable-length output records.

VLSCMP=YES
VLSCMP=NO
Default: NO
	VLSCMP
	NOVLSCMP
		whether DFSORT pads short variable-length compare fields.

VSAMEMT=YES
VSAMEMT=NO
Default: YES
	VSAMEMT
	NVSAMEMT
		whether DFSORT accepts an empty VSAM input data set.

VSAMIO=YES
VSAMIO=NO
Default: NO
	VSAMIO
	NOVSAMIO
		whether DFSORT allows a VSAM data set defined with REUSE to be sorted in-place.

ZDPRINT=YES
ZDPRINT=NO
Default: YES
	ZDPRINT
	NZDPRINT
		whether DFSORT produces printable numbers from positive summed ZD fields.

VSAM Processing

DFSORT gives you new ways to process VSAM data sets as follows:

- The new **RESET=YES** installation option and **RESET** run-time option tells DFSORT to process a VSAM output data set defined with REUSE as a NEW data set. The high-used RBA is reset to zero and the output data set is effectively treated as an initially empty cluster. RESET=YES is the IBM-supplied installation default.

  The new **RESET=NO** installation option and **NORESET** run-time option tells DFSORT to process a VSAM output data set defined with REUSE as a MOD data set. The high-used RBA is not reset and the output data set is effectively treated as an initially non-empty cluster.

- The new **VSAMEMT=YES** installation option and **VSAMEMT** run-time option tells DFSORT to accept an empty VSAM input data set and process it as having zero records. VSAMEMT=YES is the IBM-supplied installation default.

  The new **VSAMEMT=NO** installation option and **NVSAMEMT** run-time option tells DFSORT to issue an error message, terminate and give a return code of 16 if an empty VSAM data set is found.

- The new **VSAMIO=YES** installation option and **VSAMIO** run-time option tells DFSORT to allow a sort application to use the same VSAM data set for input and output, provided that RESET is in effect and the VSAM data set was defined with REUSE. The VSAM data set is processed as NEW for output and will contain the sorted input records, that is, it will be sorted in-place.

  The new **VSAMIO=NO** installation option and **NOVSAMIO** run-time option tells DFSORT to issue an error message, terminate and give a return code of 16 if the same VSAM data set is specified for input and output. VSAMIO=NO is the IBM-supplied installation default.

- DFSORT supports the VSAM extended addressability function for extended format VSAM data sets, which provides the capability of VSAM data sets larger than four gigabytes.

Processing z/OS Unix Files

DFSORTsupports files in a z/OS file system for input and output. DFSORT uses BSAM to access z/OS Unix files and is thus subject to all of the capabilities and restrictions that entails. Here's an example of a DFSORT job to sort z/OS Unix files.

```
//EXAMP   JOB ...
//SORTHFS EXEC PGM=ICEMAN
//SYSOUT  DD SYSOUT=*
//SORTIN  DD  PATH='/user/hfs.inp1.txt',PATHOPTS=ORDONLY,
//        LRECL=80,BLKSIZE=240,RECFM=FB,FILEDATA=TEXT
//        DD  PATH='/user/hfs.inp2.txt',PATHOPTS=ORDONLY,
//        LRECL=80,BLKSIZE=80,RECFM=F,FILEDATA=TEXT
//SORTOUT DD  PATH='/user/hfs.ut.txt',PATHOPTS=OWRONLY,
//        LRECL=80,BLKSIZE=80,RECFM=F,FILEDATA=TEXT
//SYSIN DD *
  SORT FIELDS=(10,8,CH,A)
/*
```

Larger Tape Block Sizes

DFSORT can use tape data sets with block sizes greater than 32760 bytes for input and output, providing improved performance and tape utilization.

The installation SDB option has been expanded to allow selection of system-determined optimum block sizes greater than 32760 bytes for output tape data sets. SDB can also be used as a run-time option. If you want to use system-determined block sizes for DASD and tape output data sets, specify one of the following values:

- **SDB=LARGE** if you want DFSORT to select tape output block sizes greater than 32760 bytes.
- **SDB=YES** or **SDB=SMALL** if you want DFSORT to select tape output block sizes up to 32760 bytes.
- **SDB=INPUT** if you want DFSORT to select tape output block sizes greater than 32760 bytes only if the tape input block size is greater than 32760 bytes. SDB=INPUT is the IBM-supplied installation default.

If you don't want DFSORT to use system-determined block sizes, specify **SDB=NO** (not recommended).

Even with SDB=LARGE or SDB=INPUT, DFSORT will not select a tape output block size greater than the BLKSZLIM in effect, so you may need to specify a value like BLKSZLIM=1G in your output DD statement. Here's an example of a DFSORT job that selects system-determined block sizes greater than 32760 bytes for SORTOUT and OUTFIL tape output data sets:

```
//EXAMP   JOB ...
//SDBOUT EXEC PGM=ICEMAN
//SYSOUT  DD SYSOUT=*
//SORTIN  DD  DSN=INPUT.DATA,DISP=SHR
//SORTOUT DD  DSN=OUTPUT1,DISP=(NEW,KEEP),UNIT=3590,VOL=SER=075834,
//            LABEL=(,SL),BLKSZLIM=1G
//OUT2    DD  DSN=OUTPUT2,DISP=(NEW,KEEP),UNIT=3590,VOL=SER=075835,
//            LABEL=(,SL),BLKSZLIM=1G
//SYSIN DD *
  OPTION SDB=LARGE
  SORT FIELDS=(10,8,CH,A)
  OUTFIL FNAMES=OUT1,OMIT=(22,3,CH,EQ,C'FLY')
/*
```

DFSORT's ICEGENER, like IEBGENER, will use the parameters SDB=LARGE, SDB=YES, SDB=SMALL, SDB=INPUT and SDB=NO if you specify them. Here's an example of an IEBGENER job that selects a system-determined block size greater than 32760 bytes for a SYSUT2 tape output data set:

```
//EXAMP   JOB ...
//SDBOUT EXEC PGM=IEBGENER,PARM='SDB=LARGE'
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD  DSN=INPUT.DATA,DISP=SHR
//SYSUT2  DD  DSN=OUTPUT3,DISP=(NEW,KEEP),UNIT=3590,VOL=SER=075836,
//            LABEL=(,SL),BLKSZLIM=1G
//SYSIN DD DUMMY
```

This same job can use DFSORT's more efficient ICEGENER facility if your site has installed ICEGENER to be invoked by the name IEBGENER. Alternatively, you can specify PGM=ICEGENER to ensure that ICEGENER is used.

Long Variable-Length Output Records

DFSORT gives you new ways to handle "long" variable-length output records. A long output record is one whose length (in the RDW) is greater than the LRECL of the SORTOUT or OUTFIL data set it is to be written to.

- The new **VLLONG=YES** installation option and **VLLONG** run-time option tells DFSORT to truncate long variable-length output records to the LRECL of the SORTOUT or OUTFIL data set. VLLONG should not be used unless you want the data at the end of long variable-length output records to be truncated for your DFSORT application; inappropriate use of VLLONG can result in unwanted loss of data.
- The new **VLLONG=NO** installation option and **NOVLLONG** run-time option tells DFSORT to issue an error message, terminate and give a return code of 16 if a long variable-length output record is found. VLLONG=NO is the IBM-supplied installation default.

Signed/Unsigned Zero

DFSORT lets you treat numeric -0 and +0 values as signed (that is, different) or unsigned (that is, the same) for collation, comparisons, editing, conversions, minimums and maximums.

- The new **SZERO=YES** installation option and **SZERO** run-time option tells DFSORT to treat numeric zero values as signed for INCLUDE, INREC, MERGE, OMIT, OUTFIL, OUTREC and SORT statement processing. -0 and +0 are treated as different values, that is, -0 is treated as a negative value and +0 is treated as a positive value. SZERO=YES is the IBM-supplied installation default.
- The new **SZERO=NO** installation option and **NOSZERO** run-time option tells DFSORT to treat numeric zero values as unsigned for INCLUDE, INREC, MERGE, OMIT, OUTFIL, OUTREC and SORT statement processing. The new **UZERO** option tells ICETOOL to treat numeric zero values as unsigned for DISPLAY, OCCUR, SELECT and UNIQUE operator processing (overriding the default of treating zero values as signed for these operators). -0 and +0 are treated as the same value, that is, -0 and +0 are both treated as positive values.

Time-of-Day Installation Options Control

DFSORT makes it easy to control the resources you allow DFSORT applications to use based on the day and time they run.

Four time-of-day installation modules (ICETD1-4) can be used independently to activate different sets of installation defaults on different days at different times. Each environment installation module (ICEAM1-4) can enable one or more time-of-day installation modules.

This capability allows new levels of control for DFSORT installation defaults. For example, you could use the ICEPRM01 member below (in PARMLIB) to set up larger DSA and TMAXLIM limits for batch program-invoked DFSORT applications that run off-shift (6:00pm-5:59am) during the week, and all weekend:

```
INV
  ENABLE=TD1
  SVC=(,ALT)
  DSA=48
TD1
  WKDAYS=(1800,559)
  WKEND=ALL
  SVC=(,ALT)
  DSA=96 TMAXLIM=8388608
```

Here's what the ICEPRM01 member does:

- INV operands

  INV changes the batch program-invoked installation defaults.

  ENABLE=TD1 enables ICETD1 for ICEAM2, that is, for batch program-invoked applications. Any or all of ICETD1-4 can be enabled for ICEAM2 in any order.

  SVC=(,ALT) specifies that the alternate SVC is to be used.

  DSA=48 sets the DSA limit to 48 megabytes, overriding the IBM-supplied default for DSA of 64 megabytes.

  The IBM-supplied defaults are used for all other installation options. For TMAXLIM, the IBM-supplied default is 6 megabytes.

- TD1 operands

TD1 changes the the first time-of-day installation defaults. ICETD1 is enabled for ICEAM2.

WKDAYS=(1800,559) specifies that ICETD1 will be activated for DFSORT applications that start on Monday through Friday between 6:00pm (1800) and 5:59am (559).

WKEND=ALL specifies that ICETD1 will be activated for DFSORT applications that start any time on Saturday or Sunday.

SVC=(,ALT) specifies that the alternate SVC is to be used.

DSA=96 sets the DSA limit to 96 megabytes, overriding the 48 megabyte limit for ICEAM2, whenever ICETD1 is activated.

TMAXLIM=8388608 sets the TMAXLIM limit to 8 megabytes, overriding the 6 megabyte limit for ICEAM2, whenever ICETD1 is activated.

The IBM-supplied defaults are used for all other installation options. However, note that TD1 can be used to override any set of installation options you like for particular day and time ranges.

By setting your installation defaults appropriately, you can fine-tune DFSORT's resource usage for special situations at your site.

More Work Data Sets

DFSORT raised the number of JCL and dynamically allocated work data sets you can use from 100 to 255. Any valid ddname of the form SORTWKdd can be used for work data sets. Of course, SORTWK00-99 can still be used. But so can ddnames like SORTWK3B, SORTWK#5 and SORTWKXY.

The new limit of 255 work data sets and the new ddnames increase significantly the amount of data you can sort in a single application. Use more work data sets only when you need them for extremely large sorts.

More Merge Data Sets

DFSORT raised the number of input data sets you can use from 16 to 100. The new limit of 100 increases significantly the amount of data you can merge in a single application.

Simplified Installation and Customization

DFSORT installation and customization is easier than ever.

All features can be installed together, and the mode of operation (resident or nonresident) can be chosen at IPL time. The number of FMIDs has been reduced from 10 to 3 and the number of libraries required to install DFSORT has been reduced from 40 to 26. These changes eliminate many decisions and speed up installation and customization.

IBM's DFSORT and DFSMS teams have simplified the process of replacing IEBGENER with DFSORT's popular ICEGENER facility. You only need to apply **DFSMS PTF UW48193** to supply an alias of "IEBGENR" for IEBGENER, and place ICEGENER with an alias of "IEBGENER" ahead of IEBGENER in the system's search order for programs. This new process removes the need to monitor IEBGENER PTFs relative to ICEGENER.

Improvements for Copy, Merge and ICEGENER Applications

Storage above 16MB virtual can be used for copy, merge and ICEGENER applications. This provides virtual storage constraint relief and may provide improved performance for these applications.

Copy and merge modules reside above 16MB virtual. This provides additional virtual storage constraint relief.

Option-in-effect messages are produced for copy and merge applications as well as for sort applications. This makes it easier to determine the options used for a particular run.

Incomplete Spanned Records

DFSORT gives you several ways to handle incomplete records in spanned data sets. The new **SPANINC** installation and run-time option allows you to specify what you want DFSORT to do if it detects incomplete spanned records, as follows:

- **SPANINC=RC0** tells DFSORT to issue an informational message, set a return code of **0** and eliminate all incomplete spanned records it detects. Valid records are recovered.
- **SPANINC=RC4** tells DFSORT to issue an informational message, set a return code of **4** and eliminate all incomplete spanned records it detects. Valid records are recovered.
- **SPANINC=RC16** tells DFSORT to issue an error message, terminate and give a return code of **16** if an incomplete spanned record is found.

LRECL Padding and Truncation

DFSORT gives you several ways to handle a SORTIN LRECL smaller than the SORTOUT LRECL (LRECL padding) and a SORTIN LRECL larger than the SORTOUT LRECL (LRECL truncation). The new **PAD** and **TRUNC** installation and run-time options allow you to specify what you want DFSORT to do when the SORTIN or SORTINnn LRECL is different from the SORTOUT LRECL, as follows:

- **PAD=RC0** tells DFSORT to issue an informational message, set a return code of **0** and continue processing when the SORTOUT LRECL is **larger** than the SORTIN or SORTINnn LRECL.
- **PAD=RC4** tells DFSORT to issue an informational message, set a return code of **4** and continue processing when the SORTOUT LRECL is **larger** than the SORTIN or SORTINnn LRECL.
- **PAD=RC16** tells DFSORT to issue an error message, terminate and give a return code of **16** if the SORTOUT LRECL is **larger** than the SORTIN or SORTINnn LRECL.
- **TRUNC=RC0** tells DFSORT to issue an informational message, set a return code of **0** and continue processing when the SORTOUT LRECL is **smaller** than the SORTIN or SORTINnn LRECL.
- **TRUNC=RC4** tells DFSORT to issue an informational message, set a return code of **4** and continue processing when the SORTOUT LRECL is **smaller** than the SORTIN or SORTINnn LRECL.
- **TRUNC=RC16** tells DFSORT to issue an error message, terminate and give a return code of **16** if the SORTOUT LRECL is **smaller** than the SORTIN or SORTINnn LRECL.