DFSORT:
Beyond Sorting

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

This paper highlights and shows examples of some of the features of DFSORT that can improve application performance and programmer productivity. Topics included are: JOINKEYS, ICEGENER, Multiple Output, ICETOOL, INCLUDE/OMIT Tests, Reformatting Features, OUTFIL Features, SMF, TOD and ETOD Date and Time Formats, Floating Sign Format, Free Form Formats, Product Connections, Symbols, National Language Support, Time-of-Day Controls, Continue or Terminate Controls, SAS Booster, BLDINDEX, Year 2000 Features, and Sources of Information. This paper reflects DFSORT features available as of **October, 2010**.

Contents

DFSORT: Beyond Sorting

Introduction

Customers of IBM's DFSORT product seem to come in two varieties: those who know about and use the many features of the product (often in ingenious ways) and those who only use the product in the most basic way. Sure, everyone knows that sorting is important, and most appreciate the performance improvements DFSORT delivers using memory objects, hiperspace, data space, striping, compression, extended addressing, DASD and tape device architecture, processor memory, processor cache, and so on. Many even go so far as to tune DFSORT to provide maximum performance for the applications in their data center using ICEPRMxx members in PARMLIB to change DFSORT's environment-specific installation options.

But those who stop there are missing out on productivity improvements that can be just as important as performance improvements. Saving CPU and elapsed time when an application runs is great; saving programmer time to write and maintain those applications represents significant cost reductions as well.

The objective of this paper is to highlight and show examples of some of the features of DFSORT that can improve application performance and programmer productivity. This paper reflects DFSORT features available as of **October, 2010**. Even programmers already acquainted with DFSORT may not be aware of all of its features such as JOINKEYS, ICEGENER, Multiple Output, the ICETOOL Data Processing and Reporting Utility, INCLUDE/OMIT Tests, Reformatting Features, OUTFIL Features, SMF, TOD and ETOD Date and Time Formats, Floating Sign Format, Free Form Formats, Symbols, National Language Support, Time-of-Day Controls, Continue or Terminate Controls, DFSORT's Performance Booster for the SAS System, BLDINDEX and Year 2000 Features.

Since in-depth discussion is not the goal of this short paper, sources for detailed information are listed at the end of the paper.

JOINKEYS: Join and Conquer

JOINKEYS is a powerful DFSORT function that helps you to perform various "join" applications on two data sets by one or more keys. You can do an inner join, full outer join, left outer join, right outer join and unpaired combinations. The two data sets can be of different types (fixed, variable, VSAM, and so on) and lengths, and have keys in different locations. You can even do cartesian joins.

For added flexibility, the records from the input data sets can be processed in a variety of ways before and after they are joined using other DFSORT control statements such as INCLUDE, OMIT, INREC, OUTREC, SUM and OUTFIL.

Here's an example of a cartesian join using JOINKEYS:

```
//CJ EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//IN1 DD DSN=... input fileA (FB/30)
//IN2 DD DSN=... input fileB (FB/16)
//SORTOUT DD DSN=... output file (FB/36)
//SYSIN DD *
  JOINKEYS F1=IN1,FIELDS=(1,10,A)
  JOINKEYS F2=IN2,FIELDS=(21,10,A)
  REFORMAT FIELDS=(F2:21,16,F1:11,20)
  OPTION COPY
/*
```

Here's an example of a match operation using JOINKEYS. It creates the following output data sets:

- OUT12: input records with keys that appear in input file1 and input file2.
- OUT1: input records with keys that only appear in input file1.
- OUT2: input records with keys that only appear in input file2.

```
//MATCH EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//IN1 DD DSN=... input File1 (FB/10)
//IN2 DD DSN=... input File2 (FB/10)
//OUT12 DD SYSOUT=* keys in File1 and File2
//OUT1 DD SYSOUT=* keys in File1 only
//OUT2 DD SYSOUT=* kesy in File2 only
//SYSIN DD *
 JOINKEYS F1=IN1,FIELDS=(1,10,A)
 JOINKEYS F2=IN2,FIELDS=(1,10,A)
 JOIN UNPAIRED,F1,F2
 REFORMAT FIELDS=(F1:1,10,F2:1,10,?)
 OPTION COPY
 OUTFIL FNAMES=OUT12,INCLUDE=(21,1,CH,EQ,C'B'),
   BUILD=(1,10)
 OUTFIL FNAMES=OUT1,INCLUDE=(21,1,CH,EQ,C'1'),
   BUILD=(1,10)
 OUTFIL FNAMES=OUT2,INCLUDE=(21,1,CH,EQ,C'2'),
   BUILD=(11,10)
/*
```

By using DFSORT's JOINKEYS, JOIN and REFORMAT statements in various ways, you can do many different types of join and match operations to "slice and dice" your data.

ICEGENER: Make IEBGENER Jobs Fly

Here's a feature that couldn't be easier to use, yet provides excellent performance improvements. DFSORT's ICEGENER facility is a replacement for IEBGENER. Most data centers probably have hundreds of IEBGENER jobs like this one:

```
//COPYIT JOB ...
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=FLY.INPUT,DISP=SHR
//SYSUT2 DD DSN=FLY.OUTPUT,DISP=OLD
//SYSIN DD DUMMY
```

To convert this to an ICEGENER job the "hard" way, change IEBGENER in the EXEC statement to ICEGENER. Yes, I said that's the hard way. The easy way is to install ICEGENER as a direct replacement for IEBGENER so that no changes to IEBGENER jobs are required! ICEGENER is then called automatically whenever IEBGENER is requested either directly or from a program.

ICEGENER uses DFSORT to process IEBGENER jobs when possible and transfers control to IEBGENER when DFSORT can't be used. Most IEBGENER jobs that use DUMMY for SYSIN can be processed by DFSORT, resulting in significant performance improvements. As an added benefit, DFSORT issues messages containing useful information such as the number of records copied and the RECFM, LRECL, and BLKSIZE of the SYSUT1 and SYSUT2 data sets.

Installing ICEGENER as a direct replacement for IEBGENER even lets RACF's IRRUT200 utility take advantage of ICEGENER's performance improvements.

Multiple Output (OUTFIL): Slash Elapsed Time, EXCPs and CPU Time

With OUTFIL, sort, merge and copy applications can create multiple output data sets containing unedited or edited records, different ranges, subsets or samples of records, reports, and so on, from a single pass over one or more input data sets.

To show how the use of OUTFIL can slash elapsed time, EXCPs and CPU time, we did laboratory measurements in a stand-alone environment for the creation of 10 and 15 sorted subset output data sets of 30MB from an input data set of 200MB. Here's the range of performance improvements we observed with DFSORT using a single sort step with OUTFIL vs DFSORT using a sort step followed by multiple copy steps (keep in mind that performance improvements will vary depending on factors such as key size, record type, number of records, processor model, region size, and so on):

- 85% to 89% elapsed time reduction
- 13% to 30% CPU time reduction
- 89% to 92% EXCP count reduction

ICETOOL: Do Complex Tasks Easily

ICETOOL, a versatile data set processing and reporting utility, provides an easy-to-use batch front-end for DFSORT. ICETOOL combines new features with previously available DFSORT features to perform complex sorting, copying, reporting, join, match and analytical tasks using multiple data sets in a single job step.

The sixteen ICETOOL operators briefly described below provide new tools for programmers:

- COPY - copies a data set to one or more output data sets. Multiple output is handled using a single pass over the input.
- COUNT - prints a message containing the count of records in a data set. Can also be used to create an output data set containing text and the count, or to set RC=12, RC=8, RC=4 or RC=0 based on the count of records in a data set (that is, empty, not empty, higher, lower, equal or not equal).
- DATASORT - sorts data records between header and trailer records in a data set to an output data set.
- DEFAULTS - prints the DFSORT installation defaults in a separate list data set.
- DISPLAY - prints the values and characters of specified numeric and character fields in a separate list data set. Simple, tailored or sectioned reports can be produced. Maximums, minimums, totals, averages and counts can be produced.
- MERGE - merges one or more data sets to one or more output data sets. Multiple output is handled using a single pass over the input.
- MODE - sets/resets scanning and error actions.
- OCCUR - prints each unique value for specified numeric and character fields, and the number of times it occurs, in a separate list data set. Simple or tailored reports can be produced. The values printed can be limited to those for which the value count meets specified criteria (that is, all duplicates, no duplicates, higher, lower or equal).

- RANGE - prints a message containing the count of values in a range (that is, higher, lower, equal or not equal) for a numeric field.
- RESIZE - Creates a larger record from multiple shorter records, or creates multiple shorter records from a larger record, that is, resizes fixed length records.
- SELECT - selects records for an output data set based on meeting criteria (that is, all duplicates, no duplicates, first, last, first duplicate, last duplicate, higher, lower or equal) for the number of times numeric or character field values occur. Records that are not selected can be saved in a separate output data set.
- SORT - sorts a data set to one or more output data sets. Multiple output is handled using a single pass over the input.
- SPLICE - splices together fields from records that have the same numeric or character field values (that is, duplicate values), but different information. Fields from two or more records can be combined to create an output record. The fields to be spliced can originate from records in different data sets, so you can use SPLICE to do various "join" and "match" operations.
- STATS - prints messages containing the minimum, maximum, average, and total of values in numeric fields.
- SUBSET - selects records from a data set based on keeping or removing header records, relative records or trailer records. Records that are not selected can be saved in a separate output data set.
- UNIQUE - prints a message containing the count of unique values in a numeric or character field.
- VERIFY - prints a message identifying each invalid value found in decimal fields.

Here's an example of the JCL and control statements for an ICETOOL job. Other ICETOOL examples can be found in *DFSORT Application Programming Guide*, in *DFSORT: Getting Started*, and in "Sources of Information: Empower Programmers".

```
//EXAMP    JOB   A492,PROGRAMMER
//TOOL     EXEC  PGM=ICETOOL
//TOOLMSG  DD    SYSOUT=A
//DFSMSG   DD    SYSOUT=A
//TOOLIN   DD    *

* Statistics from all branches
STATS FROM(ALL) ON(18,4,ZD) ON(28,6,PD) ON(22,6,PD)
* Books from VALD and WETH
SORT FROM(BKS) TO(DAPUBS,PRPUBS) USING(SPUB)
* Separate output for California and Colorado branches
SORT FROM(ALL) USING(CACO)
* California branches profit analysis
  RANGE FROM(CADASD) ON(28,6,PD) HIGHER(-1500) LOWER(+8000)
* Branches with less than 32 employees
  RANGE FROM(ALL) ON(18,4,ZD) LOWER(32)
* Print a report for the Colorado branches
DISPLAY FROM(CODASD) LIST(RPT) -
  YDDD(YD/) TITLE('Colorado Branches Report') PAGE -
  HEADER(,'City') ON(1,15,CH) -
  HEADER('Profit/Loss','(thousands)') ON(28,6,PD,E1,U12) -
  HEADER('Employees','(thousands)') ON(18,4,ZD,A1,U05) -
  BLANK BETWEEN(5) -
  TOTAL('Total') AVERAGE('Average') -
  MINIMUM('Lowest') COUNT('Number of cities')
* Print a report of books for individual publishers
 DISPLAY FROM(DAPUBS) LIST(SECTIONS) -
   TITLE('BOOKS FOR INDIVIDUAL PUBLISHERS') PAGE -
   HEADER('TITLE OF BOOK')  ON(1,35,CH) -
   HEADER('PRICE OF BOOK') ON(170,4,BI,C1,F'$') -
   BTITLE('PUBLISHER:') BREAK(106,4,CH) -
   BAVERAGE('AVERAGE FOR THIS PUBLISHER') -
   BTOTAL('TOTAL FOR THIS PUBLISHER') -
   AVERAGE('AVERAGE FOR ALL PUBLISHERS') -
   TOTAL('TOTAL FOR ALL PUBLISHERS')
```

```
* Print the count of books in use from each publisher
OCCUR FROM(BKIN) LIST(PUBCT) BLANK -
  TITLE('Books from Publishers') DATE(DMY.) -
  HEADER('Publisher') HEADER('Books Used') -
  ON(106,4,CH) ON(VALCNT,N05)
* Separate output containing records for publishers
* with more than 4 books in use
SELECT FROM(BKIN) TO(BKOUT) ON(106,4,CH) HIGHER(4)
* Reformat REGION.IN1 to T1 so it can be spliced
  COPY FROM(REGNIN1) TO(T1) USING(CTL1)
* Reformat REGION.IN2 to T1 so it can be spliced
  COPY FROM(REGNIN2) TO(T1) USING(CTL2)
* Splice records in T1 with matching ON fields
  SPLICE FROM(T1) WITHALL -
    ON(5,5,CH)   -   Region
    WITH(1,4)    -   Office
    WITH(25,4)   -   Employees
    WITH(29,10)  -   Evaluation
    TO(REGNOUT)
/*
//ALL DD DSN=A123456.SORT.BRANCH,DISP=SHR
//BKS    DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//       DD DSN=A123456.SORT.SAMPADD,DISP=SHR
//DAPUBS DD DSN=&&DSRT,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=SYSDA
//PRPUBS DD SYSOUT=A
//SPUBCNTL DD *
 SORT FIELDS=(106,4,A,1,75,A),FORMAT=CH
 INCLUDE COND=(106,4,EQ,C'VALD',OR,106,4,EQ,C'WETH'),
         FORMAT=CH
/*
//CACOCNTL DD *
  SORT FIELDS=(1,15,CH,A)
  OUTFIL FNAMES=(CADASD,CATAPE),INCLUDE=(16,2,CH,EQ,C'CA')
  OUTFIL FNAMES=(CODASD,COTAPE),INCLUDE=(16,2,CH,EQ,C'CO')
/*
//CADASD DD DSN=&&CA,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390
//CATAPE DD DSN=CA.BRANCH,UNIT=3480,VOL=SER=111111,
//  DISP=(NEW,KEEP),LABEL=(,SL)
//CODASD DD DSN=&&CO,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390
//COTAPE DD DSN=CO.BRANCH,UNIT=3480,VOL=SER=222222,
//  DISP=(NEW,KEEP),LABEL=(,SL)
//RPT DD SYSOUT=A
//SECTIONS DD SYSOUT=A
//BKIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//PUBCT DD SYSOUT=A
//BKOUT DD DSN=A123456.BOOKS1,DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(3,3)),UNIT=3390
//REGNIN1 DD DSN=A123456.REGION.IN1,DISP=SHR
//REGNIN2 DD DSN=A123456.REGION.IN2,DISP=SHR
//T1 DD DSN=&&T1,UNIT=3390,SPACE=(CYL,(5,5)),DISP=(MOD,PASS)
//REGNOUT DD DSN=A123456.REGION.OUT,DISP=(NEW,CATLG,DELETE),UNIT=3390,
//  SPACE=(TRK,(5,5))
```

```
//CTL1CNTL DD *
* Move REGION.IN1 fields to their locations for the
* output data set
  OUTREC  BUILD=(5:1,5,      Region
                 10:21,15,   Regional Director
                 39:6,15)    Headquarters
/*
//CTL2CNTL DD *
* Move REGION.IN2 fields to their locations for the
* output data set
  OUTREC  BUILD=(1:1,4,      Office
                 5:5,5,      Region
                25:10,4,     Employees
                29:14,10,    Evaluation
                53:X)
/*
```

ICETOOL can be called directly, as in the example above, or from a program using a parameter list to which ICETOOL can additionally return information for further processing.

Control Statements: Eliminate Programming

Wouldn't it be nice to manipulate the records, fields and even bits, of data sets without doing any programming? The DFSORT control statements briefly described below make it easy for you to do this:

- INCLUDE - selects records to be kept based on logical criteria.
- OMIT - selects records to be deleted based on logical criteria.
- SUM - produces a single record for each unique sort or merge key with optional field totals.
- INREC and OUTREC - specifies reformatting, find and replace operations, group operations, parsing, justifying, squeezing, past, current and future timestamps, translation, date editing, date conversion, date field arithmetic, numeric editing, numeric conversion, substitution, arithmetic operations, hexadecimal display, bit display and sequence numbers. Can build records item by item, only overlay specific parts of records, or reformat different records in different ways by specifying how items are to be applied to records that meet given criteria.
- OPTION - overrides installation defaults and supplies optional information such as the number of records to skip before sorting.
- OUTFIL - specifies multiple output data sets, subsets, sampling, repetition, reports, reformatting, find and replace operations, group operations, parsing, justifying, squeezing, past, current and future timestamps, translation, date editing, date conversion, date field arithmetic, numeric editing, numeric conversion, substitution, arithmetic operations, hexadecimal display, bit display, sequence numbers, VB to FB and FB to VB conversion, and more. Can build records item by item, only overlay specific parts of records, or reformat different records in different ways by specifying how items are to be applied to records that meet given criteria. Can update counts and totals in existing trailer records.

Here's an example showing just a few of the things you can do with these control statements:

```
//SYSIN DD *
* Skip the first three records
   OPTION SKIPREC=3
* Keep records that meet the specified field-to-field or
* field-to-constant selection criteria
   INCLUDE COND=(7,5,PD,LE,22,5,PD,OR,25,3,CH,EQ,C'J69')
* Add a sequence number to the remaining records.
   INREC OVERLAY=(73:SEQNUM,8,ZD)
* Sort the records.
   SORT FIELDS=(20,8,ZD,A,11,2,CH,D)
* Produce one record for each unique sort key with totals
* for the specified summary fields
   SUM FIELDS=(45,4,BI,8,2,PD)
* Produce output records displaying the sort and sum fields
* intermixed with constants and arithmetic operations.
   OUTREC  BUILD=(8C'0',20,8,ZD,M11,4X,11,2,4C'*',
                  45,4,BI,M11,4X,8,2,PD,M11,
                  ((32,6,PD,MUL,-50),ADD,58,3,ZD),M11,LENGTH=8,
                  73,8)
* Produce three output data sets from one pass over the input
   OUTFIL FNAMES=(OUT1,OUT2,OUT3)
/*
```

INCLUDE and OMIT Tests

The INCLUDE and OMIT statements, and OUTFIL's INCLUDE, OMIT and SAVE operands (discussed further in "OUTFIL: Subsets"), provide extremely powerful filtering capabilities for DFSORT. A single logical condition can be specified alone, or multiple logical conditions can be joined by OR or AND to form more complex tests. The use of these statements or operands to keep or delete records as needed increases programmer productivity by eliminating programming work.

INCLUDE/OMIT: Comparisons

DFSORT's comparison tests for INCLUDE/OMIT processing allow records to be kept or deleted based on comparing a field to another field, or comparing a field to a decimal, hexadecimal, character, or current, past or future date constant. DFSORT can generate constants for the current, past or future date in a variety of character string and decimal number formats. DFSORT can also compare a two-digit year date field to another two-digit year date field, or compare a two-digit year date field to a two-digit year date constant, including the current date, a past date or a future date.

Here's an example of an INCLUDE statement that compares a packed decimal date in the form P'yyyymmdd', and a character date in the form C'yyyy-ddd', to yesterday's date:

```
INCLUDE COND=(21,5,PD,GE,DATE1P-1,OR,
  3,8,CH,GE,DATE3(-)-1)
```

Records with a P'yyyymmdd' date or a C'yyyy-ddd' date greater than or equal to yesterday's date will be included in the output data set.

Here's an example of an OMIT statement that compares a ZD field to a PD field, and compares a BI field to a decimal number.

```
OMIT COND=(7,3,ZD,EQ,18,2,PD,AND,12,1,BI,EQ,+25)
```

Records in which the ZD and PD values are equal, and the BI value is 25, will be omitted from the output data set.

INCLUDE/OMIT: Substring Search Logic

DFSORT's substring comparison tests for INCLUDE/OMIT processing allow records to be kept or deleted based on searching for a constant within a field value or a field value within a constant. This capability allows multiple conditions to be replaced with a single condition. Two types of substring comparison tests are available:

- Select records based on whether a field value is found in a constant. Here's an example of an INCLUDE statement using this type of test:

  ```
  INCLUDE COND=(11,6,SS,EQ,C'HAMMER,CHISEL,SAW   ,WRENCH')
  ```

  Records with HAMMER, CHISEL, SAW or WRENCH in positions 11-16 will be included in the output data set.

  Note that the comma is used within the constant to separate the valid 6-character values; any character that will not appear in the field value can be used as a separator in the constant. SAW must be padded with three blanks on the right to make it a 6-character value so that it will be properly compared to the 6-character field in positions 11-16.

  The single condition above can replace the following four conditions:

  ```
  INCLUDE COND=(11,6,CH,EQ,C'HAMMER',OR,
                11,6,CH,EQ,C'CHISEL',OR,
                11,6,CH,EQ,C'SAW',OR,
                11,6,CH,EQ,C'WRENCH')
  ```

  Note that unlike the SS case above, DFSORT will automatically pad SAW with three blanks in this case.

  Here's another example with this type of test:

  ```
  OMIT COND=(25,3,SS,EQ,C'D05D08D12',AND,32,2,PD,GT,130)
  ```

  Records with D05, D08 or D12 in positions 25-27 and with the PD value in positions 32-33 greater than 130 will be excluded from the output data set. Note that commas are not used to separate the valid 3-character values in the constant, but could be used to improve readability.

- Select records based on whether a constant is found in a field value. Here's an example of an INCLUDE statement using this type of test:

  ```
  INCLUDE COND=(1,25000,SS,EQ,C'OK')
  ```

  Records with OK anywhere in positions 1-25000 will be included in the output data set.

INCLUDE/OMIT: Bit Logic

DFSORT's bit logic capabilities for INCLUDE/OMIT processing allow records to be kept or deleted based on the values of particular bits. Two methods for using bit logic are available:

- Bit operators (BO/ALL, BM/SOME, BZ/NONE, BNO/NOTALL, BNM/NOTSOME, BNZ/NOTNONE) select records using a hexadecimal or bit mask consisting of ones (test) and zeros (don't test). Here's an example of an OMIT statement using bit operators:

  ```
  OMIT COND=(11,2,ALL,X'1234',OR,
             21,1,BZ,B'01001100'),FORMAT=BI
  ```

  Note that hexadecimal and bit masks can be used interchangeably as can the two sets of bit operators (for example, BO/ALL and BZ/NONE). The table below shows the action that would be taken for several records:

  ```
  11,2,BI value     21,1,BI value   Action
  ---------------   -------------   --------------
  X'1234' (true)    X'4C' (false)   Omit record
  X'0204' (false)   X'40' (false)   Include record
  X'F334' (true)    X'00' (true)    Omit record
  ```

- Bit comparisons select records using a bit constant consisting of ones (test for 1), zeros (test for 0), and periods (don't test). Here's an example of an OMIT statement using bit comparison:

  ```
  INCLUDE COND=(11,1,BI,EQ,B'10...1.1')
  ```

  The table below shows the action that would be taken for several records:

11,1,BI value	Action
X'85' (true)	Include record
X'C1' (false)	Omit record
X'9F' (true)	Include record

INCLUDE/OMIT: Numeric Tests

DFSORT's numeric tests for INCLUDE/OMIT processing allow records to be kept or deleted based on whether a field has numerics (field,EQ,NUM) or non-numerics (field,NE,NUM). Three types of numeric tests are available:

- Select records based on whether a character field has numerics or non-numerics. FS or CSF is used as the field format for this type of test. Here's an example of an INCLUDE statement that tests for numerics in a character field:

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

  Records with '0'-'9' in positions 21-25 will be included in the output data set.

- Select records based on whether a zoned decimal field has numerics or non-numerics. ZD is used as the field format for this type of test. Here's an example of an OMIT statement that tests for non-numerics in a zoned decimal field:

  ```
  OMIT COND=(31,12,ZD,NE,NUM)
  ```

  Records with anything other than '0'-'9' in positions 31-41, or with anything other than '0'-'9', X'C0'-X'C9' or X'D0'-X'D9' in position 42, will be omitted from the output data set.

- Select records based on whether a packed decimal field has numerics or non-numerics. PD is used as the field format for this type of test. Here's an example of an INCLUDE statement that tests for numerics in a packed decimal field:

  ```
  INCLUDE COND=(5,6,PD,EQ,NUM)
  ```

  Records with 0-9 for each digit in positions 5-10 and F, D or C for the sign in position 10 will be included in the output data set.

INCLUDE/OMIT: Short Field Logic

DFSORT's VLSCMP option allows short INCLUDE and OMIT fields to be compared as if they were temporarily padded with binary zeros. 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. To illustrate what you can do with VLSCMP, consider the following INCLUDE statement:

```
INCLUDE COND=(6,1,CH,EQ,C'1',OR,21,5,CH,EQ,C'ABCDE')
```

and a 15-byte record with a character 1 in position 6.

- By default, DFSORT uses options NOVLSCMP and NOVLSHRT, which result in termination when the short INCLUDE field at positions 21-25 is found.
- If you use NOVLSCMP and VLSHRT, DFSORT treats the entire logical expression as false when the short INCLUDE field at positions 21-25 is found, so the record is omitted (this can be very useful in certain situations).

- But if you use VLSCMP, DFSORT temporarily pads the short INCLUDE field at positions 21-25 with binary zeros and does the two comparisons. The first comparison is true (position 6 has a character 1), so the record is included even though the second comparison involves a short field.

The use of binary zero padding with VLSCMP can make a comparison true instead of false, so you need to allow for that or even take advantage of it. For example, you could use statements like:

```
OPTION VLSCMP
INCLUDE COND=(6,1,CH,EQ,C'1',OR,21,1,BI,GT,X'08')
```

to include records with character 1 in position 6 or hex 09 to hex FF in position 21. Records shorter than 21 bytes without character 1 in position 6 are omitted because position 21 is padded with hex 00 and is thus less than hex 08.

On the other hand, statements like:

```
OPTION VLSCMP
INCLUDE COND=(6,1,CH,EQ,C'1',OR,21,1,BI,LT,X'08')
```

inadvertently include records with short fields, as well as records with character 1 in position 6 or hex 00 to hex 07 in position 21. Records shorter than 21 bytes are included because position 21 is padded with hex 00 and is thus less than hex 08. To omit the unwanted records with short fields, you can add a test of the record length like so:

```
OPTION VLSCMP
INCLUDE COND=(6,1,CH,EQ,C'1',OR,
              (1,2,BI,GE,X'0015',AND,21,1,BI,LT,X'08'))
```

Now records shorter than 21 bytes (hex 0015) without character 1 in position 6 are omitted, making the padding of position 21 for records with short fields irrelevent.

When you use VLSCMP, keep in mind that short fields are padded with binary zeros and construct your INCLUDE or OMIT statement accordingly to get the results you want.

Reformatting Features

The INREC, OUTREC and OUTFIL statements provide versatile reformatting capabilities. INREC reformats records before they are sorted, copied or merged. OUTREC reformats records after they are sorted, copied or merged. OUTFIL reformats OUTFIL records. The use of these statements, either separately or together, to manipulate the fields of records in various ways increases programmer productivity by eliminating programming work.

Reformatting: FINDREP, BUILD, OVERLAY and IFTHEN

INREC, OUTREC and OUTFIL can each be used to reformat records in one of four ways using unedited, edited or converted input fields and a variety of constants, as appropriate:

- Using FINDREP, you 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')
  ```

  Whenever 'D27', 'A52' or 'X31' is found anywhere in the input records, it will be replaced by 'INVALID' in the output records, and the remaining bytes will be shifted to the right.

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

Note: "FIELDS" can be used instead of "BUILD" for the INREC and OUTREC statements. "OUTREC" can be used instead of "BUILD" for the OUTFIL statement. You might want to start using "BUILD" instead of "FIELDS" or "OUTREC" so you don't have to remember which operand goes with which statement.

Here's an example of reformatting with OUTFIL and BUILD:

```
OUTFIL FNAMES=OUT1,
  BUILD=(21,20,5,3,PD,ADD,+20,TO=PD,LENGTH=3,6C'*',40:X)
```

The output records for the OUT1 data set will contain the following:

- In output positions 1-20, the characters from input positions 21-40.
- In output positions 21-23, a 3-byte PD value equal to +20 added to the 3-byte PD value in input positions 5-7.
- In output positions 24-29, six asterisks.
- In output positions 30-40, blanks.

- Using OVERLAY, you 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 reformatting with OUTREC and OVERLAY:

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

  The output records for the SORTOUT data set will contain the following:

  - In output positions 45-52, the characters from input positions 45-52 with any lowercase letters (a-z) translated to uppercase letters (A-Z).
  - In all other output positions, the corresponding unchanged input characters.

- Using IFTHEN clauses, you 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. You can use any INCLUDE logical expression to specify the criteria. You can use FINDREP, BUILD or OVERLAY to reformat the records that meet those criteria. You can also apply FINDREP, BUILD or OVERLAY to all records, or only to records that do not meet any of the criteria.

  Here's an example of reformatting with INREC and IFTHEN clauses:

  ```
  INREC IFTHEN=(WHEN=INIT,OVERLAY=(73:SEQNUM,8,ZD)),
    IFTHEN=(WHEN=(4,5,CH,EQ,C'TYPE1'),
       BUILD=(1,40,61:C'001',73:73,8)),
    IFTHEN=(WHEN=(4,5,CH,EQ,C'TYPE2'),
       BUILD=(1,30,61:C'002',73:73,8)),
    IFTHEN=(WHEN=NONE,BUILD=(1,50,61:C'000',73:73,8))
  ```

  The output records for the SORTOUT data set will contain the following:

  - An 8-byte ZD sequence number in output positions 73-80 (0000001 for the first record, 00000002 for the second record, and so on).
  - For input records with 'TYPE1' in positions 4-8, input positions 1-40 in output positions 1-40, '001' in output positions 61-63, the sequence number in output positions 73-80 and blanks in all other output positions.
  - For input records with 'TYPE2' in positions 4-8, input positions 1-30 in output positions 1-30, '002' in output positions 61-63, the sequence number in output positions 73-80 and blanks in all other output positions.

– For input records without 'TYPE1' or 'TYPE2' in positions 4-8, input positions 1-50 in output positions 1-50, '000' in output positions 61-63, the sequence number in output positions 73-80 and blanks in all other output positions.

Reformatting: Fields and Constants

INREC, OUTREC and OUTFIL can be used to reformat input records for output. You can define which input fields are included in the output record, in what order the input fields appear, how they are aligned and how they are edited, converted or changed. Character and hexadecimal separators can be inserted before, between and after the input fields.

Here's an example of rearranging fields and adding constants:

```
OUTREC BUILD=(5,2,5C'*',18,6,C'ABC',+30,TO=BI,LENGTH=4,X'01',
 80:X)
```

The output records for the SORTOUT data set will contain the following:

- In output positions 1-2, the characters from input positions 5-6.
- In output positions 3-7, five asterisks.
- In output positions 8-13, the characters from input positions 18-23.
- In output positions 14-16, the characters ABC.
- In output positions 17-20, the decimal number +30 converted to a 4-byte binary value (X'0000001E').
- In output position 21, the hexadecimal value 01.
- Blanks in output positions 22-80.

Reformatting: Timestamps

INREC, OUTREC and OUTFIL let you generate constants for the current date, the current time, a past date, or a future date, in a variety of character, zoned decimal and packed decimal formats, giving you an easy way to add timestamps to your output records.

Here's an example of timestamps:

```
INREC IFTHEN=(WHEN=(5,1,CH,EQ,C'A'),
  BUILD=(2X,DATE=(MD4/),X,TIME,X,1,20)),
 IFTHEN=(WHEN=(5,1,CH,EQ,C'B'),
  BUILD=(2X,DATE=(DM4-),X,TIME=(24.),X,1,20))
```

The output records for the SORTOUT data set will contain the following:

- Blanks in positions 1-2.
- In output positions 3-12, the current date in the form 'mm/dd/yyyy' (e.g. '03/28/2005') if position 5 has 'A', or in the form 'dd-mm-yyyy' (e.g. '28-03-2005') if position 5 has 'B'.
- A blank in position 13.
- In output positions 14-21, the current time in the form 'hh:mm:ss' (e.g. '11:05:31') if position 5 has 'A', or in the form 'hh.mm.ss' (e.g. '11.05.31') if position 5 has 'B'.
- A blank in position 22.
- In output positions 23-42, the characters from input positions 1-20.

Reformatting: Hexadecimal and Bit Display and Translation

INREC, OUTREC and OUTFIL let you display any field in a record as hexadecimal or bits. Here's an example of hexadecimal display and bit display:

```
OUTFIL BUILD=(1,2,TRAN=HEX,C'**',1,2,TRAN=BIT)
```

The output records for the SORTOUT data set will contain the following:

- In output positions 1-4, the hexadecimal representation (0-F) of input positions 1-2.
- Two asterisks in output positions 5-6.
- In output positions 7-22, the bit representation (0 or 1) of input positions 1-2.

You can even display an entire variable-length record, including the variable part, as hexadecimal or bits. Here's an example for hexadecimal:

```
OUTREC BUILD=(1,4,1,4,TRAN=HEX,5,TRAN=HEX)
```

The output records for the SORTOUT data set will contain the following:

- In output positions 1-4, the record descriptor word (RDW).
- In output positions 5-8, the hexadecimal representation of the RDW.
- Starting in output position 9, the hexadecimal representation of the variable bytes starting in input position 5.

If you already have a value displayed in hexadecimal (0-F), you can translate it back to binary. Here's an example:

```
INREC BUILD=(5,6,TRAN=UNHEX)
```

The output records for the SORTOUT data set will contain a binary value in positions 5-7 corresponding to the EBCDIC hex in positions 5-10 (e.g. C'ABCD12' will be translated to X'ABCD12').

If you already have a value displayed in bits (0 or 1), you can translate it back to binary. Here's an example:

```
INREC BUILD=(21,8,TRAN=UNBIT)
```

The output records for the SORTOUT data set will contain a binary value in position 21 corresponding to the EBCDIC bits in positions 21-28 (e.g. C'11000001' will be translated to B'11000001' = X'C1').

Reformatting: Translation

INREC, OUTREC and OUTFIL let you translate characters as follows:

- From ASCII to EBCDIC.

  Here's an example of ASCII to EBCDIC translation:

  ```
  OUTREC OVERLAY=(11:11,10,TRAN=ATOE)
  ```

  The ASCII characters in position 11-20 will be replaced by their equivalent EBCDIC characters.

- From EBCDIC to ASCII.

  Here's an example of EBCDIC to ASCII translation:

  ```
  OUTREC BUILD=(1,4,5,TRAN=ETOA)
  ```

  The EBCDIC characters from position 5 to the end of the VB record will be replaced by their equivalent ASCII characters.

- From lowercase EBCDIC letters to uppercase EBCDIC letters.

  Here's an example of lowercase to uppercase translation:

  ```
  INREC OVERLAY=(21:21,11,TRAN=LTOU,48:48,8,TRAN=LTOU)
  ```

  The two specified fields will be overlaid as follows (the rest of the record will remain unchanged):

  – In output positions 21-31, the characters from input positions 21-31 with any lowercase letters translated to uppercase letters (e.g. 'Vicky-123,x' will be translated to 'VICKY-123,X').

  – In output positions 48-55, the characters from input positions 48-55 with any lowercase letters translated to uppercase letters.

- From uppercase EBCDIC letters to lowercase EBCDIC letters.

  Here's an example of uppercase to lowercase translation:

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

  Any uppercase letters in the entire data portion of the variable-length record will be translated to lowercase letters (e.g. 'CARRIE-005, CA' will be translated to 'carrie-005, ca').

- According to the ALTSEQ table in effect.

  Here's an example of translation using an ALTSEQ table:

  ```
  ALTSEQ CODE=(0040,5C40)
  OUTFIL BUILD=(1,20,21,30,TRAN=ALTSEQ,C'END')
  ```

  The output records for the SORTOUT data set will contain the following:

  – In output positions 1-20, the characters from input positions 1-20.

  – In output positions 21-50, the characters from input positions 21-50 with each binary zero (X'00') and asterisk (X'5C') character translated to a blank (X'40') character.

  – In output positions 51-53, the characters 'END'.

Reformatting: Date Conversion

INREC, OUTREC and OUTFIL make it easy to convert a date field to another type, and to convert a date field to a day of the week. The date field can be in either Julian or Gregorian form, with 2-digit or 4-digit years, in CH, ZD and PD format. The day of the week can be displayed as a 1 digit, 3 character, or 9 character constant.

Here's an example of some different types of date conversions:

```
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)
```

Reformatting: Date Field Arithmetic

INREC, OUTREC and OUTFIL make it easy to perform various types of arithmetic on date fields. You can add or subtract days, months or years from a date field, calculate the number of days between two date fields, calculate the next or previous day of the week for a date field, or calculate the last day of the week, month, quarter or year for a date field. The date fields can be in either Julian or Gregorian form, with 2-digit or 4-digit years, in CH, ZD and PD format.

Here's an example of some different types of date arithmetic:

```
OPTION COPY,Y2PAST=1996
INREC BUILD=(1,6,Y2W,SUBDAYS,+7,TOJUL=Y4T,X,
  1,6,Y2W,DATEDIFF,21,7,Y4W,X,
  9,7,Y4T,NEXTDMON,TOGREG=Y4T(/))
```

Reformatting: Numeric Editing

INREC, OUTREC and OUTFIL provide sophisticated editing capabilities for controlling how numeric fields, including floating-point fields, SMF, TOD and ETOD date and time fields, and two-digit year date fields, are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on. You can edit BI, FI, PD, PD0, ZD, FS, SFF, UFF, FL, Y2x, DTn, DCn, DEn, TMn, TCn and TEn format fields. Twenty-seven pre-defined editing masks are available for commonly used numeric patterns, encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available via user-defined editing masks. Editing capabilities are particularly useful for OUTFIL reports.

Here's an example of OUTREC numeric editing:

```
OUTFIL FNAMES=(OUT1,BACKUP),BUILD=(5:21,7,ZD,M18,25:46,3,ZD,M12,
         51:8,3,PD,EDIT=(IT.TTT))
```

The output records for the OUT1 and BACKUP data sets will contain the following:

- Blanks in output positions 1-4.
- In output positions 5-14, the ZD value in input positions 21-27 converted to a pattern of SII,IIT.TT where S represents a sign of blank for plus or - for minus, I represents a leading insignificant digit and T represents a significant digit.
- Blanks in output positions 15-24.
- In output positions 25-29, the ZD value in input positions 46-48 converted to a pattern of ST.TT where S represents a sign of blank for plus or - for minus and T represents a significant digit.
- Blanks in output positions 30-50.
- In output positions 51-56, the PD value in input positions 8-10 converted to a pattern of IT.TTT where I represents a leading insignificant digit and T represents a significant digit.

Reformatting: Numeric Conversion

INREC, OUTREC and OUTFIL make it easy to convert a field in one numeric format, including floating-point fields, SMF, TOD and ETOD date and time fields, and two-digit year date fields, to another numeric format. You can convert BI, FI, PD, PD0, ZD, FS, SFF, UFF, FL, Y2x, DTn, DCn, DEn, TMn, TCn and TEn input fields to BI, FI, PD, PDC, PDF, ZD, ZDC, ZDF or FS output fields. The length of the converted field can be defaulted or specified.

Here's an example of numeric conversion:

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

The zoned decimal value in positions 21-25 of the input record is converted to a packed decimal value in positions 1-3 of the output record (with an F for a postiive sign or a D for a negative sign). The zoned decimal value in positions 8-11 of the input record is converted to a fixed-point value in positions 4-5 of the output record. .

Reformatting: Arithmetic Operations

INREC, OUTREC and OUTFIL let you perform various types of arithmetic operations using numeric fields in a record, and decimal constants. You can edit the results as discussed in "Reformatting: Numeric Editing" or convert the results as discussed in "Reformatting: Numeric Conversion".

Your arithmetic expressions can combine BI, FI, PD, PD0, ZD, FS, SFF, UFF, FL, DTn, DCn, DEn, TMn, TCn and TEn fields (p,m,f), decimal constants (+n and -n), operators (MIN, MAX, MUL, DIV, MOD, ADD and SUB), and parentheses. The result is a signed, 15-digit or 31-digit ZD number that you can edit using the available edit options (M0-M26, EDIT, SIGNS, LENGTH), or convert to BI, FI, ZD, ZDC, ZDF, PD, PDC, PDF or FS format using the available to options (TO, LENGTH).

Here's an example of arithmetic expressions:

```
INREC 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=(+,-))
```

Reformatting: Sequence Numbers

INREC, OUTREC and OUTFIL let you generate BI, PD, ZD or FS sequence numbers in the output records. Starting values and increment values can be specified or defaulted.

The sequence numbers are assigned in the order in which the records are received for INREC, OUTREC or OUTFIL processing. Note that with INREC, the sequence numbers are assigned before sorting, copying or merging, while with OUTREC, the sequence numbers are assigned after sorting, merging or copying. With OUTFIL OUTREC, the sequence numbers are assigned to each OUTFIL record, so records for different OUTFIL data sets can have the same sequence numbers.

Here's an example of sequence numbers assigned before sorting with INREC:

```
INREC OVERLAY=(73:SEQNUM,8,ZD,START=1000,INCR=100)
SORT FIELDS=(25,5,CH,A)
```

Zoned-decimal sequence numbers 00001000, 00001100, 00001200, and so on will overlay positions 73-80 of the output records. Note that an F-sign is used for zoned-decimal sequence numbers so they will be displayable and printable. Since we used INREC, the sequence numbers are in the original (unsorted order) of the input records. We could actually use another SORT on these sequence numbers to put the records back into their original input order.

Here's an example of sequence numbers assigned after sorting with OUTREC:

```
SORT FIELDS=(25,5,CH,A)
OUTREC OVERLAY=(73:SEQNUM,8,ZD,START=1000,INCR=100)
```

Since we used OUTREC, the sequence numbers are in the sorted order of the output records.

You can also restart the sequence numbers at the START value each time the value in a specified field changes. Here's an example with OUTFIL:

```
OUTFIL OVERLAY=(81:SEQNUM,5,ZD,RESTART=(5,3))
```

If the first three records had AAA in positions 5-7, they would be given sequence numbers 00001, 00002 and 00003 in positions 81-85 (the default START and INCR values are 1). If the next two records had BBB in positions 5-7, they would be given sequence numbers 00001 and 00002 (starting over at the default START value of 1).

Note that without RESTART=(5,3), the first five records would be given sequence numbers 00001, 00002, 00003, 00004 and 00005.

Reformatting: Lookup and Change

INREC, OUTREC and OUTFIL provide for selection of a character string, hexadecimal string or input field for output from a lookup table, based on a character, hexadecimal or bit string as input. This powerful lookup and change feature allows you to substitute meaningful words or phrases for cryptic values. This capability is particularly useful for generation of commands and for OUTFIL and ICETOOL reports.

Here's an example of lookup and change:

```
OUTFIL FNAMES=SHOW,
       BUILD=(31,44,6X,
         11,2,CHANGE=(7,
           C'RD',C'READ',
           C'UD',C'UPDATE',
           C'AR',112,7,
           X'FFFF',C'EMPTY'),
         NOMATCH=(C'UNKNOWN'),
         4X,
         21,1,
         CHANGE=(10,
           B'.1......',C'VSAM',
           B..0......',C'NON-VSAM'))
```

The output records for the SHOW data set will contain the following:

- Input positions 31-74 in output positions 1-44.
- Blanks in output positions 45-50.
- In output positions 51-57, READ or UPDATE if input positions 11-12 have RD or UD, respectively, input positions 112-118 if input positions 11-12 have AR, EMPTY if input positions 11-12 have hexadecimal FFFF, or UNKNOWN if input positions 11-12 do not have RD, UD, AR or hexadecimal FFFF.
- Blanks in output positions 58-61.
- In output positions 62-71, VSAM if bit 1 of position 21 is on or NON-VSAM if bit 1 of position 21 is off.

Note: IFTHEN clauses can be used for more complex lookup and change functions involving the use of any INCLUDE logical expression for lookup and any BUILD or OVERLAY items for change.

Reformatting: Justifying

INREC, OUTREC and OUTFIL let you left-justify or right-justify data within fields. Left-justifying removes leading blanks and shifts the remaining data to the left. Right-justifying removes trailing blanks and shifts the remaining data to the right. Leading and trailing strings can be inserted, and other characters can be treated as if they were blanks.

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

```
INREC OVERLAY=(31:31,20,JFY=(SHIFT=LEFT,LEAD=C'<',TRAIL=C'>'),
   61:61,10,JFY=(SHIFT=RIGHT,PREBLANK=C'+-'))
```

The data in positions 31-50 is left-justified with a '<' before the first non-blank character and a '>' after the last non-blank character.

The data in positions 61-70 is right-justified after leading and trailing '+' and '-' characters are replaced with blanks.

Reformatting: Squeezing

INREC, OUTREC and OUTFIL let you left-squeeze or right-squeeze data within fields. Left-squeezing removes all blanks and shifts the remaining data to the left. Right-squeezing removes all blanks and shifts the remaining data to the right. Leading and trailing strings can be inserted, a string can be substituted for squeezed out blanks, blanks can be kept between paired apostrophes or quotes, and other characters can be treated as if they were blanks.

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

```
OUTREC BUILD=(15,20,SQZ=(SHIFT=LEFT,MID=C',',PAIR=QUOTE),
   101,15,SQZ=(SHIFT=RIGHT,PREBLANK=C'*'))
```

The data in input positions 15-34 is placed in output positions 1-20, shifted to the left with the blanks squeezed out except where they appear between quotes, and with a comma inserted wherever blanks are squeezed out between the first and last non-blank.

The data in input positions 101-114 is placed in output positions 21-35, shifted to the right with the blanks and asterisks squeezed out.

Reformatting: Parsing Delimited Fields

INREC, OUTREC and OUTFIL let you extract delimited fields and use them with all of the functions discussed previously for fixed fields. Using PARSE or IFTHEN PARSE, and its variety of suboperands (ABSPOS, ADDPOS, SUBPOS, STARTAFT, STARTAT, ENDBEFR, ENDAT, PAIR and FIXLEN), you define the rules for extracting each of up to 100 delimited fields (for example, CSV, keyword separated fields, and many other types) into a %nn fixed parsed field (%00-%99). You can then use these %nn fields where you can use p,m fields in BUILD, OVERLAY, IFTHEN BUILD and IFTHEN OVERLAY.

You can use % to skip over delimited fields you don't need.

This very powerful DFSORT function can be used to extract delimited variable position/length data in many different forms into %nn fields, and then edit, convert, justify, squeeze, translate, lookup and change, or do arithmetic with the %nn fields.

Here's an example of parsing and reformatting comma separated values (CSV):

```
OUTREC PARSE=(%01=(ENDBEFR=C',',FIXLEN=8),
              %=(ENDBEFR=C','),
              %03=(ENDBEFR=C',',FIXLEN=5),
              %04=(FIXLEN=12)),
   BUILD=(%01,14:%03,TRAN=LTOU,25:%04,UFF,TO=FS,LENGTH=15)
```

The PARSE operand defines the rules for extracting the delimited fields from each record into %nn fixed parsed fields as follows:

- The first CSV field is extracted into the 8-byte %01 field. If the extracted data is less than 8 bytes, it is padded on the right with blanks.
- The second CSV field is ignored.
- The third CSV field is extracted into the 5-byte %03 field. If the extracted data is less than 5 bytes, it is padded on the right with blanks.
- The fourth CSV field is extracted into the 12-byte %04 field. If the extracted data is less than 12 bytes, it is padded on the right with blanks.

The BUILD operand reformats the records using the %nn fields as follows:

- Output positions 1-8 contains the data in the %01 parsed field.
- Output positions 14-18 contains the data in the %03 parsed field translated from lowercase to uppercase.
- Output positions 25-39 contains the numeric digits from the %04 parsed field converted to FS format.

Reformatting: IFTHEN Group Operations

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 sorting groups of 20-byte records.

```
INREC IFTHEN=(WHEN=GROUP,BEGIN=(1,5,CH,EQ,C'DATE:'),
  PUSH=(21:7,8))
OPTION EQUALS
SORT FIELDS=(21,8,CH,A)
OUTREC BUILD=(1,20)
```

The SORTOUT data set will contain groups of records starting with 'DATE:' in positons 1-5, sorted by the date field in positions 7-14 of the 'DATE:' record (that is, sorted by the date value in the first record of each group).

OUTFIL Features

OUTFIL is a very versatile DFSORT control statement. It allows you to create one or more output data sets for a sort, copy or merge application from a single pass over one or more input data sets. It also provides a rich set of features that can be used with one output data set or many output data sets to increase programmer productivity by eliminating programming work.

OUTFIL: ddnames

The OUTFIL operands FNAMES and FILES specify the ddnames of the output data sets. FNAMES specifies one or more full 8-byte ddnames. FILES specifies one or more 2-character suffixes (xx) for SORTOFxx ddnames. SORTOUT is used as the ddname for an OUTFIL statement without FNAMES or FILES.

Here's an example of OUTFIL ddnames:

```
OUTFIL FNAMES=OUTPUT01,STARTREC=3
OUTFIL FNAMES=(DISK,TAPE)
OUTFIL SAMPLE=20,FILES=(05,AB),FNAMES=TEMP
OUTFIL ENDREC=50
```

The first OUTFIL statement uses a ddname of OUTPUT01.

The second OUTFIL statement uses ddnames of DISK and TAPE.

The third OUTFIL statement uses ddnames of SORTOF05, SORTOFAB and TEMP.

The fourth OUTFIL statement uses a ddname of SORTOUT.

OUTFIL: Subsets

The OUTFIL operands INCLUDE, OMIT and SAVE can be used to select the records to be included in each output data set. The INCLUDE and OMIT operands provide all of the capabilities of the INCLUDE and OMIT statements discussed in "INCLUDE and OMIT Tests". SAVE can be used to select the records that are not selected for any other subset, eliminating the need to specify complex conditions.

Here's an example of OUTFIL subsets:

```
OUTFIL INCLUDE=(8,6,CH,EQ,C'ACCTNG'),FNAMES=GP1
OUTFIL INCLUDE=(8,6,CH,EQ,C'DVPMNT'),FNAMES=GP2
OUTFIL INCLUDE=(8,6,SS,EQ,C'ADMIN ,RESRCH'),FNAMES=GP3
OUTFIL SAVE,FNAMES=NOT123
```

Records with ACCTNG in positions 8-13 are written to the GP1 data set. Records with DVPMNT in positions 8-13 are written to the GP2 data set. Records with ADMIN or RESRCH in positions 8-13 are written to the GP3 data set. Records without ACCTNG, DVPMNT, ADMIN or RESRCH in positions 8-13 are written to the NOT123 data set.

OUTFIL: Reformatting and Multiple Records

The OUTFIL operands FINDREP, BUILD, OVERLAY and IFTHEN can be used to reformat the records in each output data set. The FINDREP, BUILD, OVERLAY and IFTHEN operands provide all of the reformatting capabilities discussed in "Reformatting Features".

OUTFIL BUILD also provides one capability not available with INREC BUILD or OUTREC BUILD; it lets you use /, n/ or /.../ to create many output records from each input record. You can split an input record into pieces, use the input fields in one or more of the output records, double or triple space, and so on. Here are examples of splitting an input record into pieces and triple spacing:

```
OUTFIL FNAMES=PIECES,BUILD=(1,20,/,21,60)
OUTFIL FNAMES=TSPACE,BUILD=(1,80,2/)
```

Two output records will be written to PIECES for each input record. The first record will contain input positions 1-20 followed by 40 blanks. The second record will contain input positions 21-80.

Each input record will be written to TSPACE followed by two blank records (triple-spacing).

See "OUTFIL: VB to FB Conversion" for another use of BUILD.

OUTFIL: Short Record Padding

The OUTFIL operand VLFILL allows DFSORT to continue processing if a variable-length record is too short to contain all specified OUTFIL BUILD fields. Missing bytes are replaced with the specified character or hexadecimal fill byte so the filled fields can be processed. Here's an example:

```
OUTFIL FNAMES=OUT1,BUILD=(1,4,11,20,2X,35,10),VLFILL=C' '
OUTFIL FNAMES=OUT2,BUILD=(1,4,15,5,52),VLFILL=X'FF'
```

For the OUT1 output data set, missing bytes in positions 11-30 or 35-44 of the variable-length input records will be replaced with blanks.

For the OUT2 output data set, missing bytes in positions 15-19 of the variable-length input records will be replaced with FF bytes.

See "OUTFIL: VB to FB Conversion" for another use of VLFILL.

OUTFIL: Trimming

The OUTFIL operand VLTRIM can be used to remove trailing bytes of the specified type from the end of variable-length records. DFSORT decreases the length of the record by the number of trailing trim bytes, effectively removing the trim bytes.

Here's an example of OUTFIL trimming:

```
OUTFIL VLTRIM=X'00'
```

Trailing binary zero (X'00') bytes will be removed.

See "OUTFIL: FB to VB Conversion" for another use of VLTRIM.

OUTFIL: VB to FB Conversion

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

Here's an example of OUTFIL FB to VB conversion:

```
OUTFIL FNAMES=VBFB,VTOF,BUILD=(5,14,32,8,2Z,22,6)
```

The fixed-length output records for the VBFB data set will contain positions 5-18 of the variable input records, positions 32-39 of the variable input records, two binary zeros and positions 22-27 of the variable input records.

When you specify VTOF, DFSORT automatically uses VLFILL=C' ' by default. So the OUTFIL statement above is actually equivalent to:

```
OUTFIL FNAMES=VBFB,VTOF,BUILD=(5,14,32,8,2Z,22,6),
  VLFILL=C' '
```

and if your records are short, missing bytes in positions 5-18, 32-39 or 22-27 of the variable-length input records will be replaced with blanks. You can specify VLFILL=byte explicitly if you want a fill byte other than blank with VTOF Here's an example of using asterisk as the fill byte:

```
OUTFIL FNAMES=FLR1,CONVERT,BUILD=(5,14,32,8,2Z,22,6),
  VLFILL=C'*'
```

OUTFIL: FB to VB Conversion

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

Here are some examples of FB to VB conversion:

```
OUTFIL FNAMES=FBVB1,FTOV
OUTFIL FNAMES=FBVB2,FTOV,BUILD=(1,10,C'=',21,10)
OUTFIL FNAMES=FBVB3,FTOV,
 IFTHEN=(WHEN=(5,1,BI,EQ,+5),
         OVERLAY=(12:15,3,PD,ADD,7,3,PD,TO=PD,LENGTH=3)),
 IFTHEN=(WHEN=(5,1,BI,EQ,+8),
         BUILD=(1,20,YDDD=(D4.))),
 IFTHEN=(WHEN=NONE,OVERLAY=(12:-1,TO=PD,LENGTH=3))
```

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

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

The variable-length output records for the FBVB3 data set will contain a 4-byte RDW followed by different fixed-length data depending on the value in position 5:

- If position 5 has X'05', the fixed-length data after the RDW will consist of the input record with positions 12-14 overlaid by the PD sum of input positions 15-17 and 7-9.
- If position 5 has X'08', the fixed-length data after the RDW will consist of the characters from input positions 1-20 followed by the current date in the form 'ddd.yyyy'.
- If position 5 does not contain X'05' or X'08', the fixed-length data after the RDW will consist of the input record with positions 12-14 overlaid by the PD value -1.

VLTRIM=byte can be used with FTOV to remove trailing bytes from the end of the variable-length output records. Here's an example:

```
OUTFIL FNAMES=FBVB4,FTOV,VLTRIM=C'*'
```

The variable-length output records for the FBVB4 data set will contain a 4-byte RDW followed by the fixed-length input records without any trailing asterisks.

OUTFIL: Ranges

The OUTFIL operands STARTREC, ENDREC and ACCEPT can be used to select a range of records to be included in each output data set. STARTREC starts processing at a specific input record. ENDREC ends processing at a specific input record. ACCEPT ends processing after a specific number of recods have been "accepted" (that is, not deleted).

Here's an example of OUTFIL ranges:

```
OUTFIL FNAMES=FRONT,ENDREC=500
OUTFIL FNAMES=MIDDLE,STARTREC=501,ENDREC=2205
OUTFIL FNAMES=BACK,STARTREC=2206
```

Input records 1-500 are written to the FRONT data set. Input records 501-2205 are written to the MIDDLE data set. The remaining input records are written to the BACK data set.

See "OUTFIL: Sampling" for another use of STARTREC and ENDREC.

OUTFIL: Sampling

The OUTFIL operand SAMPLE can be used to sample records in a variety of ways. The sample consists of the first m records in every nth interval. STARTREC and ENDREC 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

OUTFIL: Repetition

The OUTFIL operand REPEAT lets you write each output record multiple times. The repeated records are identical, unless the SEQNUM operand discussed in "Reformatting: Sequence Numbers" 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,BUILD=(1,80,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: Split

The OUTFIL operands SPLIT1R, SPLIT and SPLITBY can be used with the operands FNAMES and FILES to split records in various ways among multiple output data sets as follows:

- SPLIT1R=n writes the first n output records to the first output data set, the second n output records to the second data set, and so on; when each output data set has n records, any remaining records are written to the last output data, so the records in each output data set will be contiguous.

  Here's an example of OUTFIL SPLIT1R:

  ```
  OUTFIL FNAMES=(T1,T2,T3,T4),SPLIT1R=50
  ```

  Records 1-50 are written to the T1 data set, records 51-100 are written to the T2 data set, records 101-150 are written to the T3 data set, and records 151-n (where n is the last record, for example, 220) are written to the T4 data set.

Use SPLIT1R rather than SPLIT or SPLITBY if you want to ensure that each output data set has contiguous records.

- SPLIT writes the first output record to the first output data set, the second output record to the second output data set, and so on; when each output data set has one record, the rotation starts again with the first output data set, so the records in each output data set will not necessarily be contiguous.

  Here's an example of OUTFIL SPLIT:

  ```
  OUTFIL FNAMES=(PIPE1,PIPE2,PIPE3),SPLIT
  ```

  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.

  SPLIT can also be used with / in OUTFIL OUTREC to put pieces of each input record into different data sets. Here's an example:

  ```
  OUTFIL FNAMES=(PART1,PART2,PART3),SPLIT,
    BUILD=(1,20,/,21,20,/,41,20)
  ```

  Positions 1-20 of each input record are written to PART1. Positions 21-40 of each input record are written to PART2. Positions 41-60 of each input record are written to PART3.

- SPLITBY=n writes the first n output records to the first output data set, the second n output records to the second data set, and so on; when each output data set has n records, the rotation starts again with the first output data set, so the records in each output data set will not necessarily be contiguous.

  Here's an example of OUTFIL SPLITBY:

  ```
  OUTFIL FNAMES=(OUT1,OUT2),SPLITBY=50
  ```

  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 OUT1 data set, records 151-200 are written to the OUT2 data set, and so on.

OUTFIL: Update counts and totals

The OUTFIL operand IFTRAIL can be used to ensure that counts and totals in an existing trailer record reflect the actual data records in the output file.

You can use any logical expression to identify the existing trailer record. You can set up the count and total fields using DFSORT's numeric editing or numeric conversion functions.

Here's an example of updating a count and total in an existing trailer record:

```
OUTFIL IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'9'),
  TRLUPD=(11:COUNT+2=(M11,LENGTH=8),
     25:TOTAL=(15,6,ZD,TO=ZD,LENGTH=8))
```

The existing trailer record is identified by a '9' in position 1. The count of the output data records plus 2 is placed in positions 11-18 of the trailer record using the M11 edit mask. The total of the ZD values in positions 15-20 of the data records is placed in positions 25-32 using the TO=ZD conversion function. The header (first) record is not used for the count or total.

OUTFIL: Reports

The OUTFIL operands LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, NODETAIL, REMOVECC, BLKCCH1, BLKCCH2, BLKCCT1, BUILD, OVERLAY, FINDREP and IFTHEN can be used to create highly detailed three-level (report, page and section) reports containing a variety of report elements you can specify.

Report, page and section headers can contain the current date in a variety of forms, the current time in a variety of forms, edited or converted page numbers, character strings, hexadecimal strings, blank lines and input fields.

Report, page and section trailers can contain the current date in a variety of forms, the current time in a variety of forms, edited or converted page numbers, character strings, hexadecimal strings, blank lines, input fields, edited or converted record counts and edited or converted totals, maximums, minimums, and averages.

Data lines for reports can be reformatted in many ways as discussed in "OUTFIL: Reformatting and Multiple Records".

Here's an example of an OUTFIL report. The details and output for this OUTFIL report example and others can be found in *DFSORT Application Programming Guide*.

```
SORT FIELDS=(3,10,A,16,13,A),FORMAT=CH
OUTFIL FNAMES=WEST,
  INCLUDE=(42,6,CH,EQ,C'West'),
  HEADER1=(5/,18:'    Western Region',3/,
              18:'Profit and Loss Report',3/,
              18:'     for  ',&DATE,3/,
              18:'      Page',&PAGE),
  BUILD=(6:16,13,24:31,10,ZD,M5,LENGTH=20,75:X),
  SECTIONS=(3,10,SKIP=P,
    HEADER3=(2:'Division:  ',3,10,5X,'Page:',&PAGE,2/,
             6:'Branch Office',24:'       Profit/(Loss)',/,
             6:'-------------',24:'--------------------'),
TRAILER3=(6:'=============',24:'====================',/,
          6:'Total',24:TOTAL=(31,10,ZD,M5,LENGTH=20),/,
          6:'Lowest',24:MIN=(31,10,ZD,M5,LENGTH=20),/,
          6:'Highest',24:MAX=(31,10,ZD,M5,LENGTH=20),/,
          6:'Average',24:AVG=(31,10,ZD,M5,LENGTH=20),/,
          3/,2:'Average for all Branch Offices so far:',
             X,SUBAVG=(31,10,ZD,M5))),
TRAILER1=(8:'Page ',&PAGE,5X,'Date:  ',&DATE,5/,
          8:'Total Number of Branch Offices Reporting:  ',
            COUNT,2/,
          8:'Summary of Profit/(Loss) for all',
            ' Western Division Branch Offices',2/,
          12:'Total:',
              22:TOTAL=(31,10,ZD,M5,LENGTH=20),/,
          12:'Lowest:',
              22:MIN=(31,10,ZD,M5,LENGTH=20),/,
          12:'Highest:',
              22:MAX=(31,10,ZD,M5,LENGTH=20),/,
          12:'Average:',
              22:AVG=(31,10,ZD,M5,LENGTH=20))
```

Creating Reports: ICETOOL vs OUTFIL

Both the OUTFIL statement and ICETOOL's DISPLAY operator can be used to create reports. While each performs some reporting functions that the other does not, in general the difference between them is one of control and effort. With OUTFIL, you have more control over the appearance of reports, but considerable effort may be required on your part since you have to specify in detail where every piece of information is to appear in the report and how it is to look. With ICETOOL, much of the work of determining where information is to appear in the report and how it is to look can be done for you, but you have somewhat less control over the appearance of the reports.

ICETOOL should be your primary choice for creating reports since it is easier to use than OUTFIL. But remember that OUTFIL is available if you need any of its specific reporting features, or more control of the appearance of reports than ICETOOL gives you.

SMF, TOD and ETOD Date and Time Formats

DFSORT and ICETOOL allow the use of a wide variety of character and numeric formats. DFSORT's SMF date formats (DT1, DT2 and DT3), TOD date formats (DC1, DC2 and DC3), ETOD date formats (DE1, DE2 and DE3), SMF time formats (TM1, TM2, TM3 and TM4), TOD time formats (TC1, TC2, TC3 and TC4) and ETOD time formats (TE1, TE2, TE3 and TE4) allow INREC, OUTREC, OUTFIL and ICETOOL's DISPLAY and OCCUR operators to display the normally unreadable SMF, TOD (STCK) and ETOD (STCKE) date and time values in a wide range of recognizable ways.

Here's an example of SMF date and time values edited for readability by DISPLAY:

```
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 data set will contain SMF date values in the form 'yyyy/mm/dd' and SMF time values in the form 'hh:mm:ss'.

Here's an example of TOD date and time values edited for readability by OUTREC:

```
OUTREC BUILD=(1,4,
              X,21,8,DC3,EDIT=(TTTT-TTT),
              X,21,8,TC4,EDIT=(TT:TT:TT.TT))
```

The SORTOUT data set will contain TOD date values in the form 'yyyy-ddd' and TOD time values in the form 'hh:mm:ss.xx'.

Floating Sign Format

DFSORT and ICETOOL allow the use of a wide variety of character and numeric formats. DFSORT's floating sign format (FS) handles many data patterns that other formats cannot process.

Note: CSF can be used instead of FS for the floating sign format.

FS format allows SORT, MERGE, INCLUDE, OMIT, INREC, OUTREC, OUTFIL and various ICETOOL operators to process data values with leading zeros specified or suppressed, with a blank for positive values, with a floating plus sign for positive values, and so on.

FS format is well suited for processing data created by many types of FORTRAN, PL/1 and COBOL output formats. For example, the following FORTRAN output data values can be processed as floating sign data, illustrating the flexibility of this format:

```
      I4    SP,I4    S,I4.3    SPi,I4.3
    ----    -----    ------    --------
       5       +5       005        +005
    -247     -247      -247        -247
      25      +25       025        +025
     800     +800       800        +800
      -4       -4      -004        -004
```

Free Form Formats

DFSORT and ICETOOL allow the use of a wide variety of character and numeric formats. DFSORT's new signed free form format (SFF) and unsigned free form format (UFF) handle many data patterns that other formats cannot process. The SFF and UFF formats allow SORT, MERGE, INCLUDE, OMIT, INREC, OUTREC, OUTFIL and various ICETOOL operators to extract numeric values (sign and digits) from any field containing digits (0-9) and any other characters.

SFF format is well suited for processing signed numeric data with decimal points, separators, leading or trailing signs, leading, trailing or embedded blanks, floating currency signs, and so on. SFF treats the extracted value as negative if a '-' or ')' character is found anywhere in the field; otherwise it treats the extracted value as positive.

Here's an example of numeric values extracted with SFF:

```
Input value              Extracted value
--------------------     --------------
     358,272,300.10         +35827230010
              (0.05)                   -5
      18,272,300.12          +1827230012
               2.60                 +260
       3,100,300.72           +310030072
         (82,316.90)            -8231690
             (52.73)               -5273
```

Here's an example of an OUTFIL statement to total these input values using SFF:

```
 OUTFIL FNAMES=SHOWTOT,NODETAIL,REMOVECC,
    TRAILER1=('Total is',TOT=(1,20,SFF,M5,LENGTH=20))
```

SHOWTOT would have:

```
Total is    (336,982,066.34)
```

UFF format is well suited for processing unsigned numeric data with decimal points, separators, hyphens, parentheses, leading, trailing or embedded blanks, floating currency signs, and so on. UFF treats the extracted value as positive.

Here's an example of numeric values extracted with UFF:

```
Input value              Extracted value
--------------------     --------------
        472-381-0650          4723810650
     @@319.816.5041           3198165041
   (402)-125-3721XXX          4021253721
(317)8327691                  3178327691
     438,721,9853             4387219853
  402  125  0021               4021250021
```

Here's an example of a SORT statement to sort these input values and left-align them as '(ddd)-ddd-dddd' values using UFF:

```
SORT FIELDS=(1,20,UFF,A)
OUTREC OVERLAY=(1,20,UFF,EDIT=((TTT)-TTT-TTTT),6X)
```

Product Connections

Eliminate programming work when DFSORT control statements can do the job for your programs that call DFSORT (for example, COBOL programs that use the SORT verb, or PL/I programs that use the PLISRTx routines). Pass control statements to DFSORT like this:

```
//DFSPARM DD *
* Only include records in which:
*   the specified character field is 'T1' and
*   the specified floating sign field value is greater than -25.
  INCLUDE COND=(21,2,CH,EQ,C'T1',AND,5,4,FS,GT,-25)
/*
```

Whether DFSORT is called directly or from a program, it always uses control statements in DFSPARM. (DFSPARM is the supplied default ddname; it can be changed by the installation option PARMDDN.) This technique can also be used to provide an OPTION statement to override DFSORT installation options, or DFSORT run-time options passed by other products.

Symbols: Use Names for Fields, Constants and Output Columns

A symbol is a name (preferably something meaningful) you can use to represent a field, a constant or an output column. Sets of symbols, also called mappings, can be used to describe a group of related fields, constants and output columns, such as the information in a particular type of record. Such mappings allow you to refer to fields, constants and output columns by their symbols, freeing you from having to know the position, length and format of a field, the value of a constant or the position of an output column you want to use.

DFSORT's symbol processing feature gives you a powerful, simple and flexible way to create symbol mappings for your own frequently used data. In addition, you can obtain IBM-created symbol mappings and sample jobs for data associated with RACF, DFSMSrmm and DCOLLECT as follows:

- 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/

Symbols turn DFSORT's syntax into a high level language. Symbols can help to standardize your DFSORT applications and increase your productivity. Once you create or obtain symbol mappings, you can use the symbols they define anywhere you can use a field, constant or output column 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.

Here's an example of the JCL and control statements for an ICETOOL job that uses symbols. A complete explanation of this example can be found in *DFSORT: Getting Started*. A complete explanation of DFSORT's Symbols feature can be found in *DFSORT Application Programming Guide*.

```
//SYM2     JOB   A492,PROGRAMMER
//TOOL     EXEC  PGM=ICETOOL,REGION=1024K
//STEPLIB  DD    DSN=A492.SM,DISP=SHR
//TOOLMSG  DD    SYSOUT=A
//DFSMSG   DD    SYSOUT=A
//SYMNAMES DD    DSN=DSN=A123456.SORT.SYMBOLS,DISP=SHR
//SYMNOUT  DD    SYSOUT=*
//IN       DD    DSN=A123456.SORT.SAMPIN,DISP=SHR
//OUT      DD    DSN=A123456.SORT.SAMPOUT,DISP=SHR
//TOOLIN   DD    *
  SORT FROM(IN) TO(OUT) USING(CTL1)
  RANGE FROM(OUT) ON(Price) LOWER(Discount)
  RANGE FROM(OUT) ON(Price) HIGHER(Premium)
//CTL1CNTL DD *
  INCLUDE COND=(Course_Number,EQ,Beginning_Economics,OR,
                Course_Number,EQ,Advanced_Sociology)
  SORT FIELDS=(Title,A,
               Instructor_Last_Name,A,Instructor_Initials,A,
               Price,A)
```

National Language Support: Go Global

Do you need to sort, merge and create subsets of data according to a defined country/cultural specification and create reports using the preferred date, time and numeric notations of individual countries? DFSORT has features that can meet all of these requirements.

DFSORT uses the IEEE POSIX and X/Open global locale model, which describes language, country, and cultural information, defining the rules so DFSORT can correctly collate (SORT and MERGE) and compare (INCLUDE and OMIT) your data according to your national/cultural needs. DFSORT allows the selection of an active locale at installation or run time with the LOCALE operand and will sort, merge and select subsets of records according to the collating rules defined in the active locale. For example, specifying:

```
//DFSPARM DD *
  OPTION LOCALE=FR_CA
  SORT FIELDS=(1,20,CH,A,25,1,BI,D,30,10,CH,A)
  INCLUDE COND=(40,6,CH,EQ,50,6,CH)
/*
```

causes the CH fields in the SORT and INCLUDE statements to be processed according to the collating rules defined in locale FR_CA, the locale for the French language and the cultural conventions of Canada.

Both ICETOOL and OUTFIL allow date, time and numeric values in reports to be formatted in many of the notations used throughout the world. Examples can be found in *DFSORT Application Programming Guide*.

Time-of-Day Controls: Fine-Tune Your Options

Would you like to have DFSORT use extra resources to speed-up jobs you run off-shift and/or on weekends? DFSORT makes this easy to do.

You're probably familiar with DFSORT's ICEAM1-4 modules that let you set your default installation options for four different environments using ICEPRMxx members in PARMLIB. But did you know that you can activate four different sets of installation options based on time-of-day with DFSORT's ICETD1-4 modules? You can use any or all of ICETD1-4 for any or all of ICEAM1-4. So you could, for example, raise the DSA value from the default value of 64 (megabytes) to a larger value of 80 (megabytes) for DFSORT jobs that start off-shift and on weekends. Here's an ICEPRM01 member in PARMLIB you could activate to make those changes:

```
JCL
  ENABLE=TD1
  SVC=(,ALT)
INV
  ENABLE=TD1
  SVC=(,ALT)
TD1
  WKDAYS=(1800,559)
  WKEND=ALL
  SVC=(,ALT)
  DSA=80
```

DFSORT batch jobs (JCL, INV) that start on Monday-Friday (WKDAYS) between 6:00pm (1800) and 5:59am (559) or on weekends (WKEND=ALL) will use the ICETD1 installation defaults (for example, DSA=80) instead of the ICEAM1 or ICEAM2 defaults (for example, the shipped default of DSA=64).

By setting your ICEAM1-4 and ICETD1-4 defaults appropriately, you can fine-tune DFSORT's resource usage for special situations at your site. For complete information on ICEAM1-4 and ICETD1-4, and ICEPRMxx members, see *DFSORT Installation and Customization.*

Continue or Terminate Controls: Decide for Yourself

DFSORT lets you decide whether to terminate or continue for certain situations. You can set an installation option to tell DFSORT what to do for the majority of your jobs and use a run-time option to change your mind for specfic jobs. Here are the relevent options and the choices you can make:

- SPANINC - tells DFSORT how to handle incomplete records in spanned data sets. The choices are to terminate and set a return code of 16, or recover the valid records and set a return code of 0 or 4.
- OVFLO - tells DFSORT how to handle an overflow condition for SUM fields. The choices are to terminate and set a return code of 16, or leave the overflowing records unsummed and set a return code of 0 or 4.
- PAD - tells DFSORT how to handle an input LRECL smaller than the output LRECL. The choices are to terminate and set a return code of 16, or continue processing and set a return code of 0 or 4.
- TRUNC - tells DFSORT how to handle an input LRECL larger than the output LRECL. The choices are to terminate and set a return code of 16, or continue processing and set a return code of 0 or 4.
- GNPAD - tells DFSORT how to handle an input LRECL smaller than the output LRECL for ICEGENER jobs (installation option only). The choices are to transfer control to IEBGENER, or use DFSORT copy and set a return code of 0 or 4.

- GNTRUNC - tells DFSORT how to handle an input LRECL larger than the output LRECL for ICEGENER jobs (installation option only). The choices are to transfer control to IEBGENER, or use DFSORT copy and set a return code of 0 or 4.

For complete information on these installation options, see *DFSORT Installation and Customization*. For information on the run-time options, see *DFSORT Application Programming Guide*.

The SAS System: Boost Performance

Do you want to improve performance for SAS applications without making any changes to them? The DFSORT and SAS groups, working together, developed DFSORT's Performance Booster for The SAS System to do just that. The booster works in conjunction with SAS to process SAS sorting applications fast, significantly reducing CPU time. To take advantage of this new feature, contact SAS Institute Inc. for details of the support they provide to enable this joint enhancement.

BLDINDEX: Sort Indexes in a Flash

Do you want dramatic performance improvements for IDCAMS BLDINDEX jobs that seem to take forever? IBM's DFSMS and DFSORT groups worked together on a method that allows unchanged BLDINDEX jobs to use DFSORT automatically. Those long running BLDINDEX jobs start to run faster immediately using DFSORT. BLDINDEX's use of this new DFSORT support also allows sorting of indexes in an all-DFSMS environment. (If the available sort product does not have support for BLDINDEX equivalent to DFSORT's, BLDINDEX continues to use its own less efficient sort.)

To show the dramatic improvement possible for very large indexes with the BLDINDEX/DFSORT connection, we did laboratory measurements in a stand-alone environment for an index with 500,000 records and a combined prime and alternate key length of 220 bytes. Here's the range of performance improvements we observed using the BLDINDEX support in DFSORT R13 (keep in mind that performance improvements will vary depending on factors such as key size, number of records, processor model, region size, and so on):

- 28% to 75% elapsed time reduction
- 29% to 70% CPU time reduction
- 34% to 82% EXCP count reduction

Year 2000 Features: For 2000 and Beyond

DFSORT provides powerful Year 2000 features that allow you to sort, merge, compare, and transform character, zoned decimal and packed decimal dates with two-digit years according to a specified sliding or fixed century window.

You can handle two-digit year date fields in the following ways:

- Set the appropriate century window for your applications and use it to interpret the years (yy) correctly when you sort, merge, compare or transform two-digit year dates. For example, set a century window of 1950-2049 or 1985-2084.
- Order character, zoned decimal or packed decimal two-digit year dates with the SORT and MERGE statements. For example, order 980622 (representing June 6th, 1998) before 000622 (representing June 6th, 2000) in ascending sequence, or order 000622 before 980622 in descending sequence.

  DFSORT's full date formats (Y2T, Y2U, Y2V, Y2W, Y2X and Y2Y) make it easy to sort or merge any type of yyx...x or x...xyy date (for example, yyq, yymm, yyddd, yymmdd, qyy, mmyy, dddyy or mmddyy). DFSORT's

year formats (Y2C, Y2Z, Y2S, Y2P, Y2D and Y2B) are available if you need to sort or merge special dates (for example, Z'ddmmyy' or C'yy/ddd') or year fields (yy).

- Select records by comparing character, zoned decimal or packed decimal two-digit year date fields to date constants or other date fields with the INCLUDE and OMIT statements or OUTFIL's INCLUDE and OMIT operands. For example, select records with a date field between January 1st, 1998 and December 31st, 2003.

  DFSORT's full date formats make it easy to compare any type of yyx...x or x...xyy date field to a date constant or another date field. DFSORT's year formats are avalable if you need to compare years.

- Transform character, zoned decimal or packed decimal two-digit year dates to character four-digit year dates with or without separators, or transform packed decimal two-digit year dates to packed decimal four-digit year dates, with the INREC, OUTREC or OUTFIL statements. For example, transform P'05015' to C'2005015', C'2005/015' or P'2005015'.

  DFSORT's full date formats make it easy to transform any type of yyx...x or x...xyy date. DFSORT's year formats are available if you need to transform special dates or year fields.

In addition, you can use DFSORT's Year 2000 features with COBOL, either automatically with COBOL MLE or explicitly without MLE.

These DFSORT enhancements allow you to continue to use two-digit year dates for sorting, merging and comparing, and help you to change from using two-digit year dates to using four-digit year dates as appropriate.

The SORT2000 paper contains complete details of DFSORT's Year 2000 features. "Sources of Information" later in this document tells you how to obtain the SORT2000 paper.

Sources of Information: Empower Programmers

Many programmers I've talked to use DFSORT without referencing the DFSORT books, or the additional sources of information and examples, that IBM makes easily available. Perhaps this explains why they are often unaware of the full capabilities of DFSORT.

DFSORT home page

The DFSORT home page on the World Wide Web contains tips, techniques, examples, tricks, papers and other helpful and timely information about DFSORT. Visit the DFSORT home page at URL:

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

The DFSORT Library

To get the most out of DFSORT, every programmer who uses it should be familiar with the following books in the DFSORT library:

- *z/OS DFSORT: Getting Started (SC26-7527-05)* is an excellent tutorial about the basics of using DFSORT control statements, ICETOOL operators and Symbols. The material is taught using numerous examples which can be run using sample data sets shipped with DFSORT.
- *z/OS DFSORT Application Programming Guide (SC26-7523-05)* is a complete reference guide for the functions and options of DFSORT and DFSORT's ICETOOL, and includes many examples.
- *z/OS DFSORT Messages, Codes and Diagnosis Guide (SC26-7525-05)* can help you eliminate common sources of errors, interpret informational (ICEnnnI) and error (ICEnnnA) messages, and diagnose program failures.

- *z/OS DFSORT Tuning Guide (SC26-7526-02)* provides valuable information about tuning DFSORT and using it for effective Application Development.

Programmers who install DFSORT should use *z/OS DFSORT Installation and Customization (SC26-7524-03)* as a reference source.

Online DFSORT Books

Online books provide capabilities (e.g. search) not provided by traditional books. All of the DFSORT books can be accessed online in one of two ways:

- On the World Wide Web. You can access all of the DFSORT books by clicking the **Publications** link on the DFSORT home page at URL:

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

  Bookmark them for faster access.

- Install the displayable softcopy books, shipped on CD-ROM in the *z/OS elements and features CD Collection Kit*.

LookAt

You can use IBM's LookAt facility to access information on a specific DFSORT message by typing in its message number (e.g. ICE283A). LookAt is on the World Wide Web at:
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/

Other IBM Books

The following are some of the IBM books that contain information on using DFSORT and ICETOOL with other IBM products:

- *z/OS DFSMSrmm Reporting (SC26-7406)*
- *z/OS DFSMShsm Data Recovery Scenarios (GC35-0419)*
- *z/OS DFSMShsm Storage Administration Guide (SC35-0421)*
- *z/OS SecureWay Security Server RACF Auditor's Guide (SA22-7684)*
- *z/OS SecureWay Security Server RACF Security Administrator's Guide (SA22-7683)*
- *z/OS SecureWay Security Server RACF System Programmer's Guide (SA22-7681)*
- *z/OS MVS System Management Facilities (SMF) (SA22-7630)*
- *z/OS DFSMS Introduction (SC26-7397)*
- *z/OS DFSMS Access Method Services for Catalogs (SC26-7394)*
- *z/OS DFSMS: Using the Volume Mount Analyzer (SC26-7413)*
- *z/OS DCE Administration Guide (SC24-5904)*
- *z/OS Language Environment Programming Guide (SA22-7561)*
- *DB2 Universal Database for OS/390 and z/OS Utility Guide and Reference (SC26-9945)*

Product Tape

DFSORT provides a set of sample jobs that demonstrate techniques of interest to Storage Administrators and others who analyze data collected from DFSMShsm, DFSMSrmm, DCOLLECT and SMF. These sample jobs can be found in the ICESTGEX member of the SICESAMP library (contact your System Programmer for details). You can also access these sample jobs from the DFSORT home page. These sample jobs show some of the many ways DFSORT features such as ICETOOL and OUTFIL can be used to analyze data and generate reports:

- DCOLEX1 - DCOLLECT Example 1: VSAM report
- DCOLEX2 - DCOLLECT Example 2: Conversion reports
- DCOLEX3 - DCOLLECT Example 3: Capacity planning analysis and reports
- DFHSMEX1 - DFHSM Example 1: Deciphering Activity Logs
- DFHSMEX2 - DFHSM Example 2: Recover a DFHSM CDS with a broken index
- RMMEX1 - DFSMSrmm Example 1: SMF audit report
- RMMEX2 - DFSMSrmm Example 2: Create ADDVOLUME commands

Papers

This paper is one of a set of papers that can help programmers be more productive with DFSORT/ICETOOL. You can obtain these papers as indicated below.

All of the papers (except RACFICE2) are available for online viewing by clicking the **Product details** link on the DFSORT home page at URL:

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

- SORTUGPH

  z/OS DFSORT V1R10 PTF UK90025 and z/OS DFSORT V1R12 PTF UK90026, first available in October, 2010, provide important enhancements to DFSORT and DFSORT's ICETOOL for resizing records (RESIZE operator); updating the trailer record (IFTRAIL); processing subsets (ACCEPT); translation of ASCII to EBCDIC (TRAN=ATOE), EBCDIC to ASCII (TRAN=ETOA), EBCDIC hex to binary (TRAN=UNHEX), and more; date field arithmetic (ADDDAYS, ADDMONS, ADDYEARS, SUBDAYS, SUBMONS, SUBYEARS, DATEDIFF, NEXTDday, PREVDday, LASTDAYW, LASTDAYM, LASTDAYQ and LASTDAYY); timestamp constant with microseconds (DATE5); group functions (KEYBEGIN); use of SET and PROC symbols in control statements (JPn"string" in EXEC PARM); more information in reports; larger fields; easier migration from other sort products, and more.

  This paper highlights, describes and shows examples of the new features provided by these PTFs for DFSORT and for DFSORT's powerful, multi-purpose ICETOOL utility. It also details new and changed messages associated with these PTFs.

  You can obtain a PDF version of SORTUGPH from the DFSORT FTP site at:

  ```
  ftp.software.ibm.com/storage/dfsort/mvs/
  ```

- SORTUGPG

  z/OS DFSORT V1R5 PTF UK51706 and z/OS DFSORT V1R10 PTF UK51707, first available in November, 2009, provide important enhancements to DFSORT and DFSORT's ICETOOL for various types of two file join applications (JOINKEYS, JOIN, REFORMAT, JKFROM); date field conversions (Y2x, Y4x, TOJUL, TOGREG, WEEKDAY, DT, DTNS); merge operations (MERGE operator); alternate ddnames for merge files (MERGEIN); easier migration from other sort products, and more.

This paper highlights, describes and shows examples of the new features provided by these PTFs for DFSORT and for DFSORT's powerful, multi-purpose ICETOOL utility. It also details new and changed messages associated with these PTFs.

You can obtain a PDF version of SORTUGPG from the DFSORT FTP site at:

```
ftp.software.ibm.com/storage/dfsort/mvs/
```

- SORTUGPF

  z/OS DFSORT V1R5 PTF UK90013, first available in July, 2008, provides important enhancements to DFSORT and DFSORT's ICETOOL for ICETOOL for find and replace (FINDREP), group operations (WHEN=GROUP); sorting data between headers and trailers (DATASORT); keeping or removing the first n records, last n records and/or specific relative records (SUBSET); selecting the first n duplicate records (SELECT with FIRST(n) and FIRSTDUP(n)); splicing with non-blank fields (SPLICE with WITHANY); displaying and writing counts (DISPLAY with COUNT, EDCOUNT, BCOUNT and EDBCOUNT, and COUNT with ADD, SUB, WRITE, TEXT, DIGITS, EDCOUNT and WIDTH); reports with multiple and multipart titles (DISPLAY and OCCUR with TITLE, TLEFT and TFIRST); reports without carriage control characters (DISPLAY and OCCUR with NOCC); additional defaults (BLKSIZE for DUMMY, SKIP=0L for SECTIONS, and SORTOUT=ddname for FNAMES); easier migration from other sort products, and more. This paper highlights, describes and shows examples of the new features provided by this PTF for DFSORT and for DFSORT's powerful, multi-purpose ICETOOL utility. It also details new and changed messages associated with this PTF.

  You can obtain a PDF version of SORTUGPF from the DFSORT FTP site at:

  ```
  ftp.software.ibm.com/storage/dfsort/mvs/
  ```

- SORTPEUG

  z/OS DFSORT V1R5 PTF UK90007 and DFSORT Release 14 PTF UK90006, first available in April, 2006, provide important enhancements to DFSORT and DFSORT's ICETOOL for 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 and future date constants; testing for numerics and non-numerics; displaying hexadecimal floating-point values as integers; splitting files contiguously; suppressing page ejects in reports; using symbols for output columns; using system symbols (e.g. &SYSPLEX) in symbol constants; reformatting records before selecting or splicing; sorting and merging with larger PD and ZD fields; easier migration from other sort products, and more. This paper highlights, describes and shows examples of the new features provided by these PTFs for DFSORT and for DFSORT's powerful, multi-purpose ICETOOL utility. It also details new and changed messages associated with these PTFs.

  You can obtain a PDF version of SORTPEUG from the DFSORT FTP site at:

  ```
  ftp.software.ibm.com/storage/dfsort/mvs/
  ```

- SORTPDUG

  z/OS DFSORT V1R5 PTF UQ95214 and DFSORT R14 PTF UQ95213, first available in December, 2004, provide important enhancements to DFSORT and DFSORT's ICETOOL for conditionally reformatting records; overlaying only selected parts of records; larger numeric fields and constants; new data formats for extracting digits, and displaying TOD and ETOD date and time values; restarting sequence numbers; join and match operations; date and time constants; conversion of statistical and count values; reports; easier migration from other sort products, and more. This paper highlights, describes, and shows examples of the new features provided by these PTFs for DFSORT and for DFSORT's powerful, multi-purpose ICETOOL utility. It also details new and changed messages associated with these PTFs.

  You can obtain PDF, postscript and text versions of SORTPDUG from the DFSORT FTP site at:

  ```
  ftp.software.ibm.com/storage/dfsort/mvs/
  ```

- SORTNEW

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

You can obtain a PDF version of SORTNEW from the DFSORT FTP site at:

```
ftp.software.ibm.com/storage/dfsort/mvs/
```

- SORTTOOL

  SORTTOOL is a mini-user guide for DFSORT's versatile ICETOOL data processing and reporting utility. The major features of ICETOOL, including its JCL and control statements, are discussed at length using many examples. The objective is to show you how to use DFSORT's ICETOOL to accomplish complex tasks.

  You can obtain a PDF version of SORTTOOL from the DFSORT FTP site at:

  ```
  ftp.software.ibm.com/storage/dfsort/mvs/
  ```

- RACFICE2

  Effective security management requires flexible analysis and reporting tools. With OS/390 Version 2 Release 8, RACF introduced the RACFICE reports in 'SYS1.SAMPLIB(IRRICE)'. RACFICE used DFSORT's ICETOOL to create a set of 30+ reports based on the output of the RACF Database Unload Utility (IRRDBU00) and the RACF SMF Unload Utility (IRRADU00).

  Since then, there have been several major enhancements to DFSORT. This package, RACFICE2, demonstrates how those enhancements can be used to do even more with the output of IRRDBU00 and IRRADU00.

  In addition, the package contains a set of DFSORT Symbols that define the IRRDBU00 and IRRADU00 fields as documented in the "RACF Macros and Interfaces (SA22-7682)".

  You can obtain RACFICE2 at URL:

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

- SORT2000

  DFSORT has Year 2000 features you can use to sort, merge, compare and transform a wide variety of dates with two-digit years according to a specified sliding or fixed century window.

  SORT2000 contains a detailed discussion of DFSORT's Year 2000 features and examples of the control statements needed to order, compare and transform all kinds of character, zoned decimal and packed decimal date fields.

  You can obtain PDF, postscript and text versions of SORT2000 from the DFSORT FTP site at:

  ```
  ftp.software.ibm.com/storage/dfsort/mvs/
  ```

Summary: Take Action

The features of DFSORT described in this paper can be used to improve application performance and programmer productivity. To take advantage of all DFSORT has to offer, ensure that you have z/OS DFSORT V1R10 PTF UK90025 or z/OS DFSORT V1R12 PTF UK90026 (October, 2010) installed, obtain the listed documentation and papers or access them online, and use DFSORT's many features to go beyond sorting.