User Guide for DFSORT PTFs UK90025 and UK90026

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 is the documentation for z/OS DFSORT V1R10 PTF **UK90025** and z/OS DFSORT V1R12 PTF **UK90026**, which were first made available in **October, 2010**.

These PTFs 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.

Contents

User Guide for DFSORT PTFs UK90025 and UK90026

Introduction

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

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

DFSORT includes the versatile ICETOOL utility and the high-performance ICEGENER facility.

z/OS DFSORT V1R10 PTF **UK90025** and z/OS DFSORT V1R12 PTF **UK90026**, which were first made 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 access all of the DFSORT books online by clicking the **Publications** link on the DFSORT home page at URL:

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

This paper provides the documentation you need to start using the features and messages associated with z/OS DFSORT V1R10 PTF UK90025 or z/OS DFSORT V1R12 PTF UK90026. The information in this paper will be included in the z/OS DFSORT books at a later date.

You should refer to *z/OS DFSORT Application Programming Guide (SC26-7523-05)* for general information on DFSORT and ICETOOL features, and in particular for the framework of existing DFSORT features upon which these new features are built. You should refer to *z/OS DFSORT Messages, Codes and Diagnosis Guide (SC26-7525-05)* for general information on DFSORT messages.

Determining if this PTF is installed

If you see the following in the messages from a DFSORT run:

```
ICE201I H RECORD TYPE ...
```

the H indicates you have z/OS DFSORT V1R10 PTF UK90025 or z/OS DFSORT V1R12 PTF UK90026, so you can use the new functions described in this paper. If you do not see ICE201I H, ask your System Programmer to install z/OS DFSORT V1R10 PTF UK90025 or z/OS DFSORT V1R12 PTF UK90026 so you can use the new functions.

Summary of Changes

RESIZE

RESIZE is a new operator of ICETOOL that allows you to create a larger record from multiple shorter records, or multiple shorter records from a larger record. RESIZE makes it easy to "resize" fixed length records; you just indicate the desired length of the output records and DFSORT automatically resizes the records based on the input length.

Update the trailer record with OUTFIL

IFTRAIL is a new OUTFIL option that allows you to update count and total values in an existing trailer (last) record based on the current data records.

Various options of IFTRAIL allow you to identify the trailer record (TRLID), indicate the count and total values to be updated in the trailer record (TRLUPD), and indicate if the first record is a header record (HD=YES).

IFTRAIL makes it easy to update count and total values in an existing trailer record when you add, delete or modify data records.

DFSORT symbols can be used for fields and constants in TRLID. DFSORT symbols can be used for columns and fields in TRLUPD.

Date Field Arithmetic

New options for BUILD and OVERLAY allow you to perform various types of arithmetic on date fields in either Julian or Gregorian form, with 2-digit or 4-digit years, in CH, ZD and PD format.

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

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

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

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

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

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

DFSORT symbols can be used for fields and constants in date arithmetic.

SET and PROC symbols in control statements

SET and PROC symbols can now be included in DFSORT symbols, which in turn can be used in DFSORT and ICETOOL control statements. Up to ten special symbols (JP0-JP9), which can incorporate SET and PROC symbols, as well as other text and system symbols, can be included in the EXEC PARM string when DFSORT is called directly from JCL (PGM=SORT or PGM=ICEMAN) or ICETOOL is called directly from JCL

(PGM=ICETOOL). These JPn symbols will be treated as if they were specified in the SYMNAMES data set. This makes it easy to use DFSORT symbols incorporating SET and PROC symbols in DFSORT and ICETOOL control statements.

Accept n records with OUTFIL

ACCEPT=n is a new OUTFIL option that allows you to limit the number of input records accepted for OUTFIL processing. A record is "accepted" if it is not deleted by STARTREC, ENDREC, SAMPLE, INCLUDE or OMIT processing. ACCEPT makes it easy to limit the number of OUTFIL input records selected for processing.

New translation functions

New TRAN=keyword options in BUILD and OVERLAY provide additional character translation capabilities as follows: ASCII characters to EBCDIC characters (TRAN=ATOE); EBCDIC characters to ASCII characters (TRAN=ETOA); binary values to EBCDIC hexadecimal values (TRAN=HEX); EBCDIC hexadecimal values to binary values (TRAN=UNHEX); binary values to EBCDIC bit values (TRAN=BIT) and EBCDIC bit values to binary values (TRAN=UNBIT). This makes it easy to do such translations on specific fields or entire records.

DFSORT symbols can be used for fields in translation functions.

Begin group when key changes

KEYBEGIN=(p,m) is a new option for WHEN=GROUP clauses that allows a new group to begin whenever a specified key value changes. KEYBEGIN operates in a similar way to the existing BEGIN option, but allows the change in a specified key field to be used as the trigger instead of a logical expression. KEYBEGIN makes it easy to treat records with the same key as a group and propagate fields, identifiers and sequence numbers to the records in each group.

DFSORT symbols can be used for fields in KEYBEGIN.

Larger header/trailer fields for OUTFIL

OUTFIL's HEADERx and TRAILERx options now allow you to specify unedited fields (p,m) up to 32752 bytes (the previous limit was 256 bytes). This makes it easy to use one large header or trailer field rather than multiple header or trailer fields.

RC8 and RC12 for COUNT

ICETOOL's COUNT operator now allows you to use a new RC8 option to set RC=8 (instead of RC=12) or RC=0 based on the count of records in a data set. RC12 can also be used to set RC=12 explicitly instead of by default. RC4, RC8 and RC12 make it easy to set the return code you need for a COUNT operation.

More fields for DISPLAY

ICETOOL's DISPLAY operator now allows you to specify up to 50 HEADER and ON fields (the previous limit was 20). This makes it easy to produce reports that display more information.

Larger fields for ICETOOL

ICETOOL's DISPLAY, OCCUR, SELECT, SPLICE and UNIQUE operators now allow you to specify character ON fields of up to 4000 bytes (the previous limit was 1500 bytes). ICETOOL's DISPLAY and OCCUR operators now allow you to specify hexadecimal ON fields of up to 2000 bytes (the previous limit was 1000 bytes). This makes it easy to use one ON operand for large fields rather than multiple ON operands.

Longer reports for DISPLAY/OCCUR

ICETOOL's DISPLAY and OCCUR operators now allow you to produce reports with a line length up to 8192 bytes (the previous limit was 2048 bytes). This makes it easy to produce reports that display more information.

Timestamp with microseconds

A new DATE5 option can be used to generate a timestamp constant of the form 'yyyy-mm-dd-hh.mm.ss.nnnnnn' with the year, month, day, hours, minutes, seconds and microseconds for the run. This is similar to the previously available DATE4 constant, but includes microseconds (nnnnnn).

Operational Changes that may Require User Action

The following are operational changes that may require user action for existing DFSORT/ICETOOL applications that use certain functions as specified:

- New reserved words for symbols

  The following are new DFSORT/ICETOOL reserved words which are no longer allowed as symbols: DATE5, ADDDAYS, SUBDAYS, ADDMONS, SUBMONS, ADDYEARS, SUBYEARS, DATEDIFF, LASTDAYW, LASTDAYM, LASTDAYQ, LASTDAYY, NEXTDday and PREVDday (where day is SUN, MON, TUE, WED, THU, FRI or SAT).

  If you used any of these words as a symbol previously, you must change them. For example, if you used SUBDAYS, you can change it to Subdays.

RESIZE

Introduction

RESIZE is a new operator of ICETOOL that allows you to create a larger record from multiple shorter records, or multiple shorter records from a larger record. RESIZE makes it easy to "resize" fixed length records; you just indicate the desired length of the output records and DFSORT automatically resizes the records based on the input length.

Various options of RESIZE allow you to define the ddname for the input and output data sets, the length of the output records, and DFSORT control statements to be used for the RESIZE operation.

As an example, you could use the following RESIZE operator to create an output file with RECFM=FB and LRECL=300 from an input file with RECFM=FB and LRECL=60. Each group of five 60-byte input records will be combined into one 300-byte output record.

```
RESIZE FROM(IN) TO(OUT) TOLEN(300)
```

Syntax

The syntax for the RESIZE operator is as follows:

```
RESIZE FROM(indd) TO(outdd) TOLEN(n) USING(xxxx)
```

Detailed Description

RESIZE produces fixed length output records, with the specified TOLEN length, from fixed length input records of a different length, as follows:

- If the input length is smaller than the requested TOLEN size, multiple smaller input records are combined into one larger output record of size TOLEN. If the combined input records do not completely fill up an output record, the output record will be padded on the right with blanks. For example, if we have five input records of 20 bytes and TOLEN is 70, two output records of 70 bytes will be created. The first output record will have input records 1-3 (60 bytes) followed by 10 blanks, and the second output record will have input records 4-5 (40 bytes) followed by 30 blanks. Note that the first 10 bytes of input record 4 are not used for the first output record; **bytes or words are not "wrapped" between records.**

  To illustrate, if the following RESIZE operator was specified:

  ```
  RESIZE FROM(FB20) TO(FB70) TOLEN(70)
  ```

  and FB20 had these 20 byte input records:

  ```
  <111111111111111111>
  <222222222222222222>
  <333333333333333333>
  <444444444444444444>
  <555555555555555555>
  ```

  FB70 would have these 70 byte output records:

  ```
  <111111111111111111><222222222222222222><333333333333333333>
  <444444444444444444><555555555555555555>
  ```

- If the input length is larger than the requested TOLEN size, each larger input record is broken up into multiple smaller output records of size TOLEN. If a broken up input record does not completely fill up the multiple output records, the last of the multiple output records will be padded on the right with blanks. For example, if we have two input records of 50 bytes and TOLEN is 20, six output records of 20 bytes each will be created. The first two output records will have bytes 1-20 and 21-40 of the first input record (40 bytes), respectively. The third output record will have bytes 41-50 of the first input record (10 bytes) followed by 10 blanks. Likewise, the fourth output record will have bytes 1-20 of the second input record, the fifth output record will have bytes 21-40 of the second input record, and the sixth output record will have bytes 41-50 of the second input record followed by 10 blanks.

  To illustrate, if the following RESIZE operator was specified:

  ```
  RESIZE FROM(F50) TO(F20) TOLEN(20)
  ```

  and F50 had these 50 byte input records:

  ```
  <111111111111111111><222222222222222222><33333333>
  <444444444444444444><555555555555555555><66666666>
  ```

  F20 would have these 20 byte output records:

  ```
  <111111111111111111>
  <222222222222222222>
  <33333333>
  <444444444444444444>
  <555555555555555555>
  <66666666>
  ```

You must specify the FROM(indd), TO(outdd) and TOLEN(n) operands. The FROM data set must be fixed-length (for example, RECFM=FB). The TO data set must also be fixed-length (unless you use an OUTFIL statement with the FTOV operand to change the fixed-length resized records to variable-length output records). If a VSAM input data set is used, it will be treated as TYPE=F (fixed-length) by default. If you specify a variable-length input data set (or use TYPE=V for a VSAM input data set), the RESIZE operation will be terminated.

The USING(xxxx) operand is optional; do not supply your own MODS or OUTREC statement.

DFSORT is called to copy or sort the indd data set, as appropriate, before the records are resized. ICETOOL uses its E35 exit to create a larger record from multiple smaller records, or to create multiple smaller records from a larger record. ICETOOL passes the EQUALS option to DFSORT to ensure that if records are sorted, duplicate records are kept in their original input order when resized.

If USING(xxxx) is specified, any SORT, INCLUDE, OMIT, INREC, or SUM statement specified in xxxxCNTL is processed **before** the records are resized so the order of the records, and the input length, will be affected by these control statements. Any OUTFIL statements specified in xxxxCNTL are processed **after** the records are resized.

When ICETOOL is called using the parameter list interface, the 1-byte operation status indicator in the Return Area will be set to 0 or 4 for a RESIZE operator in the same way as for the existing operators. No operation specific values are returned for RESIZE.

The operands described below can be specified in any order:

- **FROM(indd)**

  Specifies the ddname of the input data set to be read by DFSORT for this operation. An indd DD statement must be present.

- **TO(outdd)**

  Specifies the ddname of the output data set to be written by DFSORT for this operation. An outdd DD statement must be present. The ddname specified in the TO operand must not be the same as the ddname specified in the FROM operand.

- **TOLEN(n)**

  Specifies the record length you want ICETOOL to use for the resized output records. n can be 1 to 32760. n must not be equal to the input record length.

- **USING(xxxx)**

  Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx. If USING(xxxx) is specified, an xxxxCNTL DD statement must be present, and the control statements in it must conform to the rules for DFSORT's SORTCNTL data set.

  If you want to override dynamic allocation of work data sets for this operation, you can use xxxxWKdd DD statements for that purpose.

Example 1

This example illustrates how you can create larger records from smaller records.

```
RESIZE FROM(IN1) TO(OUT1) TOLEN(40)
```

The IN1 data set has RECFM=FB and LRECL=10 with these 10-byte records:

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

The OUT1 data set has RECFM=FB and LRECL=40 with these 40 byte records:

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

Example 2

```
RESIZE FROM(OLD) TO(NEW) TOLEN(15) USING(CTL1)
//CTL1CNTL DD *
  OMIT COND=(2,4,ZD,EQ,0)
  SORT FIELDS=(1,1,CH,A)
/*
```

This example illustrates how you can create larger records from a subset of smaller sorted records.

The OLD data set has RECFM=FB and LRECL=5 with these 5-byte records:

```
C0005
B0000
A0008
I1234
F0053
D0123
H0001
G0000
E0022
```

The NEW data set will have RECFM=FB and LRECL=15 with these 15-byte records:

```
A0008C0005D0123
E0022F0053H0001
I1234
```

Note that before the records were resized, the two records with 0 in positions 2-5 were omitted, and the remaining records were sorted as directed by the DFSORT control statements in CTL1CNTL. The last output record was padded with blanks on the right to 15 bytes.

Example 3

```
RESIZE FROM(IN3) TO(OUT3) TOLEN(3) USING(CTL1)
//CTL1CNTL DD *
  OUTFIL FNAMES=OUT3,OMIT=(1,3,CH,EQ,C' '),OVERLAY=(10:X)
/*
```

This example illustrates how you can break up large records into multiple smaller records.

The IN3 data set has RECFM=FB and LRECL=18 with these 18-byte records:

```
000111222333444555
666777888999
```

Every 3-byte field in each large IN3 record will be broken up into a single 3-byte field and then padded on the right with blanks to 10-bytes. TOLEN(3) indicates that the resized records will have a length of 3 bytes. OVERLAY=(10:X) expands each resized record to 10 bytes in OUT3. OMIT=(1,3,CH,EQ,C' ') removes any resized records that are completely blank (that is, the two blank resized records resulting from the blanks in positions 13-18 of the second input record).

The OUT3 data set will have RECFM=FB and LRECL=10 with these 10-byte records:

```
000
111
222
333
444
555
666
777
888
999
```

Update the trailer record with OUTFIL

Introduction

IFTRAIL is a new option for the existing OUTFIL statement. IFTRAIL allows you to update count and total values in an existing trailer (last) record to reflect the actual data records in an OUTFIL data set. Whereas TRAILER1 lets you build a new trailer record, IFTRAIL lets you update an existing trailer record using the COUNT and TOTAL features of TRAILER1. When you add, delete or modify input data records to create an OUTFIL data set, you can use IFTRAIL to ensure that the trailer record has accurate updated count and total values.

The count and total values are determined using the data records processed as input to OUTFIL. If slash (/) is used in OUTFIL BUILD to produce multiple output data records, the count and total values are determined using the data records processed as input to OUTFIL. For example, if you use OUTFIL BUILD=(1,80,/,1,80) to create 6 OUTFIL output records from 3 OUTFIL input records, the count is 3, not 6.

You must identify the trailer record using a unique condition (for example, 'TRL' in positions 1-3). You then specify how count and total values in the identified trailer record are to be updated. The trailer record will NOT be treated as a data record, that is, no other OUTFIL processing will be performed against the trailer record. You can optionally specify that the header (first) record will NOT be treated as a data record.

As a simple example, you could use the following OUTFIL statement to write the header (first) record, data records with 'D1', and trailer record (identified by '9' in position 1) in the OUTFIL data set, and set an accurate count and total in the trailer record for the included input data records.

```
  OUTFIL FNAMES=OUT1,INCLUDE=(10,2,CH,EQ,C'D1'),
   IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'9'),
   TRLUPD=(11:COUNT=(M11,LENGTH=8),
          25:TOTAL=(15,6,ZD,EDIT=(IIIIIT))))
```

The trailer record will **not** be treated as a data record; it will not be subject to INCLUDE processing, and will not be used for the updated count or total values.

Since HD=YES is specified, the header record will **not** be treated as a data record; it will not be subject to INCLUDE processing, and will not be used for the updated count or total values.

Syntax

The syntax for the IFTRAIL operand of the OUTFIL statement is:

```
IFTRAIL=(TRLID=(logexp),TRLUPD=(c:item,...),HD=YES)
```

Detailed Description

IFTRAIL allows you to update count and total fields in the trailer record of the data sets for an OUTFIL group.

You can use IFTRAIL to identify a trailer record and indicate count and total fields to be updated in that record. The identified trailer record will **not** be used to determine the count or totals. If you specify HD=YES, the header (first) record will **not** be used to determine the count or totals.

You cannot specify IFTRAIL with any of the following operands in an OUTFIL statement: HEADER1, TRAILER1, HEADER2, TRAILER2, NODETAIL, SECTIONS, LINES, SPLIT, SPLITBY, SPLIT1R, REPEAT, FTOV, VTOF, VLTRIM or VLFILL.

You must specify the TRLID and TRLUPD operands. The HD=YES operand is optional.

- **TRLID=(logexp)**

  Specifies the criteria to be tested to determine if a record is the trailer record. When a record is found that meets the criteria, it will be treated as the last record, and subsequent records will not be processed for the OUTFIL group. Thus if you use TRLID=(1,1,CH,EQ,C'9') to identify the trailer record and you have three records with '9' in position 1, the first record with '9' in position 1 will be treated as the trailer record, and records after that will not appear in the OUTFIL data sets.

  See the discussion of INCLUDE statement in *z/OS DFSORT Application Programming Guide (SC26-7523-05)* for details of the logical expressions that you can use. You can specify all of the logical expressions for TRLID in the same way that you can specify them for the INCLUDE statement, except that:

  – You cannot specify FORMAT=f

  – You cannot specify D2 format

  – Locale processing is not used

  – VLSCMP and VLSHRT are not used. The trailer record must be long enough to contain all fields in the logical expression. Trailer record checking will be skipped for any variable-length record that is too short to contain a field in the TRLID logical expression.

  The trailer record will not be treated as a data record; all other OUTFIL processing (for example, INCLUDE, OMIT, BUILD, OVERLAY, and so on) will be bypassed for the trailer record. The updated count and total values will not include the trailer record.

  DFSORT symbols can be used for fields and constants in the logical expression in the same way they can be used for the INCLUDE statement.

- **TRLUPD=(c:item,...)**

  Specifies the position where each count or total is to be updated in the trailer record.

  For fixed-length records, the first input and output data byte starts at position 1. The trailer record will be truncated or padded with blanks to the length of the data records, if appropriate. You cannot change the length of the trailer record using TRLUPD.

  For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4. The original length of the trailer record will not be changed for output. You cannot change the length of the trailer record using TRLUPD. If you specify an item beyond the end of the trailer record, it will not be updated.

c: specifies the output position (column) in the trailer record to be updated. If you do not specify c: for the first item, it defaults to 1:. If you do not specify c: for any other item, it starts after the previous item. You must specify items in ascending output column order, and an item must not overlap a previous item in the trailer record. For a variable-length trailer record, you must not specify columns 1-4.

item specifies the count or total you want to update. Any of the following count and total items can be used as described for TRAILER1 in *z/OS DFSORT Application Programming Guide (SC26-7523-05)*:

- COUNT=(edit)
- COUNT=(to)
- COUNT+n=(edit)
- COUNT+n=(to)
- COUNT-n=(edit)
- COUNT-n=(to)
- TOTAL=(p,m,f,edit)
- TOTAL=(p,m,f,to)
- TOT=(p,m,f,edit)
- TOT=(p,m,f,to)

The count and total values are determined using the data records processed as input to OUTFIL.

If TOT or TOTAL is used, the specified p,m,f field from each data record will be totalled, even if the trailer record is not found. OUTFIL BUILD, OVERLAY or IFTHEN processing does NOT affect the values that are totalled; the values in the original OUTFIL input records are used for the totals.

An invalid field amount can result in a data exception (0C7 ABEND) or incorrect numeric output. Be careful to use HD=YES if the header record does not have valid total fields. For example, if the input file has these records:

```
H***
0001
0002
T***
```

and this IFTRAIL operand is specified:

```
  IFTRAIL=(TRLID=(1,1,CH,EQ,C'T'),
    TRLUPD=(5:TOT=(1,4,ZD,TO=ZD,LENGTH=5))
```

an 0C7 ABEND will result when DFSORT attempts to total the invalid 1,4,ZD field in the header (H***) record. The use of HD=YES would prevent this.

An invalid total field can also be encountered if the trailer record is not found due to erroneous TRLID conditions. For example, if the input file has these records:

```
H***
0001
0002
T***
```

and this IFTRAIL operand is specified:

```
  IFTRAIL=(TRLID=(1,2,CH,EQ,C'TR'),
    TRLUPD=(5:TOT=(1,4,ZD,TO=ZD,LENGTH=5))
```

the last (T***) record will not be identified as the trailer record since it does not have 'TR' in positions 1-2. Thus, an 0C7 ABEND will result when DFSORT attempts to total the invalid 1,4,ZD field in the last (T***) record. The use of the correct condition for identifying the trailer (T***) record would prevent this.

If slash (/) is used in OUTFIL BUILD to produce multiple output data records, the count and total values are determined using the data records processed as input to OUTFIL. For example, if you use OUTFIL BUILD=(1,80,/,1,80) to create 6 OUTFIL output records from 3 OUTFIL input records, the count is 3, not 6.

DFSORT symbols can be used for columns and fields in TRLUPD.

- **HD=YES**

  Specify HD=YES if you want the first record treated as a header record rather than as a data record. With HD=YES, OUTFIL processing (for example, INCLUDE, OMIT, BUILD, OVERLAY, and so on) will be bypassed for the header record. The updated count and total values will not include the header record.

  If HD=YES is specified, the TRLID test will not be applied to the first record.

Example 1

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

This example illustrates how you can split an input file with a header, and a trailer containing a count and total for the input data records, into two output files each of which has the header and a trailer with accurate count and total for the included data records.

The SORTIN data set has RECFM=FB and LRECL=80 with these input records:

```
H 2010/07/06
D key1   0100
D key2   0200
D key2   0118
D key1   0150
D key1   0025
D key2   1000
D key2   0310
T X  0007 QR 001903  D52-007-321-7526
```

The first record is a header record with a date. The records with D in position 1 are data records; they have key1 or key2 in positions 3-6 and an amount field in positions 10-13.

The trailer record is identified by a 'T' in position 1. Positions 6-9 contain a count (0007) of the data records, and positions 14-19 contain a total (001903) for the amount fields in the data records. In addition, there are other static fields in the trailer record that must not be changed. Note that IFTRAIL with HD=YES does not process the header or trailer records as data records, so they are not included in the count or total.

The first OUTFIL statement writes the header record, data records with key1, and trailer record, in the OUT1 data set. The IFTRAIL operand is used to identify the trailer (TRLID), indicate the first record is a header (HD=YES), and update the count and total in the trailer to reflect the input data records included in this output data set (TRLUPD).

OUT1 will have these records:

```
H 2010/07/06
D key1   0100
D key1   0150
D key1   0025
T X  0003 QR 000275  D52-007-321-7526
```

The second OUTFIL statement writes the header record, data records with key2, and trailer record, in the OUT2 data set. The IFTRAIL operand is used to identify the trailer (TRLID), indicate the first record is a header (HD=YES), and update the count and total in the trailer to reflect the input data records included in this output data set (TRLUPD). Note that IFTRAIL with HD=YES does not process the header or trailer records as data records, so they are not included in the count or total.

OUT2 will have these records:

```
H 2010/07/06
D key2   0200
D key2   0118
D key2   1000
D key2   0310
T X  0004 QR 001628  D52-007-321-7526
```

Example 2

```
  SORT FIELDS=(5,3,CH,A)
  OUTFIL IFTRAIL=(TRLID=(5,3,CH,EQ,C'999'),
    TRLUPD=(34:COUNT+1=(EDIT=(IIIT))))
```

This example illustrates how you can sort a VB input file and correct an out-of-date count in the trailer record.

The SORTIN data set has RECFM=VB and LRECL=40 with these input records:

```
Len|Data
14  D52  Frank
13  D51  Marc
16  E07  William
15  C21  Martin
17  H05  Sri Hari
37  999  2010-07-01  2010-07-06     3
```

There is no header record. The trailer record is identified by '999' in positions 5-7 and is supposed to have the count of all of the records (data+trailer) in positions 34-37. In this case, since the input file has 5 data records and a trailer record, the count should be 6 but it is 3 which is incorrect. So we use the DFSORT statements to SORT the records and correct the count. Note that IFTRAIL without HD=YES processes the first record as a data record rather than as a header record. IFTRAIL does not process the trailer record as a data record, so it is not included in the count. Since we want the trailer record included in the count, we use COUNT+1 in TRLUPD.

SORTOUT will have these records:

```
Len|Data
15  C21  Martin
13  D51  Marc
14  D52  Frank
16  E07  William
17  H05  Sri Hari
37  999  2010-07-01  2010-07-06     6
```

Example 3

```
  SORT FIELDS=(1,1,CH,A,17,2,CH,A,3,12,CH,A)
  OUTFIL OMIT=(1,1,CH,EQ,C'0'),
    IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'9'),
      TRLUPD=(9:COUNT=(M11,LENGTH=4),
             22:TOT=(21,5,SFF,EDIT=(STT.TT),SIGNS=(+,-)),
             37:TOT=(28,5,SFF,EDIT=(STT.TT),SIGNS=(+,-))))
```

This example illustrates how you can sort three input files each with a header and trailer, and create an output data set with one header record, the sorted data records, and one trailer record with count and totals for all of the data records in the new output data set.

The SORTIN data set consists of these three concatenated input files, each with RECFM=FB and LRECL=50:

Input file1

```
0 27-80273-5218
1 SAN JOSE      CA  -1.23  +0.35
1 PALO ALTO     CA  +8.34  +1.23
1 DALLAS        TX  -0.03  -3.41
1 LOS ANGELES   CA  +7.63  -7.02
1 NEW YORK      NY  -2.12  +5.31
9 COUNT=0005  TOTAL1=+12.59  TOTAL2=-03.54
```

Input file2

```
0 27-80273-5218
1 MORGAN HILL   CA  -3.01  -1.07
1 ARMONK        NY  +6.22  +1.52
1 DETROIT       MI  +3.05  -5.61
9 COUNT=0003  TOTAL1=+06.26  TOTAL2=-05.16
```

Input file3

```
0 27-80273-5218
1 BUFFALO       NY  -8.05  +3.83
1 GILROY        CA  +2.75  +1.97
9 COUNT=0002  TOTAL1=-05.30  TOTAL2=+05.80
```

After the three concatenated input files are sorted, the records will look like this:

```
0 27-80273-5218
0 27-80273-5218
0 27-80273-5218
1 GILROY        CA  +2.75  +1.97
1 LOS ANGELES   CA  +7.63  -7.02
1 MORGAN HILL   CA  -3.01  -1.07
1 PALO ALTO     CA  +8.34  +1.23
1 SAN JOSE      CA  -1.23  +0.35
1 DETROIT       MI  +3.05  -5.61
1 ARMONK        NY  +6.22  +1.52
1 BUFFALO       NY  -8.05  +3.83
1 NEW YORK      NY  -2.12  +5.31
1 DALLAS        TX  -0.03  -3.41
9 COUNT=0002  TOTAL1=-05.30  TOTAL2=+05.80
9 COUNT=0003  TOTAL1=+06.26  TOTAL2=-05.16
9 COUNT=0005  TOTAL1=+12.59  TOTAL2=-03.54
```

Note that the three 0 records have been sorted to the beginning. Since HD=YES is specified in IFTRAIL, the first 0 record will be treated as a header record, whereas the second and third 0 records will be treated as data records. By using OMIT=(1,1,CH,EQ,C'0') on the OUTFIL statement, we remove the second and third 0 records, so we end up with only one 0 (header) record for output.

Note that the three 9 records have been sorted to the end. IFTRAIL with TRLID=(1,1,CH,EQ,C'9') identifies the first 9 record as the trailer record so the first 9 record is updated with the correct count and totals. IFTRAIL does not process records after the trailer (first 9) record, so we end up with only the updated (first 9) trailer record for output. The second and third 9 records are not used for output.

SORTOUT contains these output records:

```
0 27-80273-5218
1 GILROY        CA  +2.75  +1.97
1 LOS ANGELES   CA  +7.63  -7.02
1 MORGAN HILL   CA  -3.01  -1.07
1 PALO ALTO     CA  +8.34  +1.23
1 SAN JOSE      CA  -1.23  +0.35
1 DETROIT       MI  +3.05  -5.61
1 ARMONK        NY  +6.22  +1.52
1 BUFFALO       NY  -8.05  +3.83
1 NEW YORK      NY  -2.12  +5.31
1 DALLAS        TX  -0.03  -3.41
9 COUNT=0010  TOTAL1=+13.55  TOTAL2=-02.90
```

The output file has only one header record and one trailer record, and the trailer record has the correct count and totals for the data records.

As an alternative, you could use these ICETOOL control statements with the same concatenated input (CON) to produce the same output (OUT):

```
//TOOLIN DD *
DATASORT FROM(CON) TO(OUT) HEADER TRAILER USING(CTL1)
/*
//CTL1CNTL DD *
  OMIT COND=(1,1,SS,EQ,C'09')
  SORT FIELDS=(17,2,CH,A,3,12,CH,A)
  OUTFIL FNAMES=OUT,
    IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'9'),
      TRLUPD=(9:COUNT=(M11,LENGTH=4),
             22:TOT=(21,5,SFF,EDIT=(STT.TT),SIGNS=(+,-)),
             37:TOT=(28,5,SFF,EDIT=(STT.TT),SIGNS=(+,-))))
/*
```

Example 4

```
//TOOLIN DD *
DATASORT FROM(CON) TO(OUT2) HEADER TRAILER USING(CTL1)
/*
//CTL1CNTL DD *
  OMIT COND=(1,1,SS,EQ,C'09')
  INREC OVERLAY=(51:X)
  SORT FIELDS=(51,1,CH,A)
  OUTFIL FNAMES=OUT2,BUILD=(1,50),
    IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'9'),
      TRLUPD=(9:COUNT=(M11,LENGTH=4),
             22:TOT=(21,5,SFF,EDIT=(STT.TT),SIGNS=(+,-)),
             37:TOT=(28,5,SFF,EDIT=(STT.TT),SIGNS=(+,-))))
/*
```

This example is similar to Example 3, but illustrates how to concatenate the input records and keep them in their original order for output. The input files are the same. However, in this case, we use an INREC statement to add a blank at the end of each record. We can then SORT on that blank. Since DATASORT has EQUALS in effect, the data records will be sorted in their original order.

OUT2 will have these records:

```
0 27-80273-5218
1 SAN JOSE      CA  -1.23  +0.35
1 PALO ALTO     CA  +8.34  +1.23
1 DALLAS        TX  -0.03  -3.41
1 LOS ANGELES   CA  +7.63  -7.02
1 NEW YORK      NY  -2.12  +5.31
1 MORGAN HILL   CA  -3.01  -1.07
1 ARMONK        NY  +6.22  +1.52
1 DETROIT       MI  +3.05  -5.61
1 BUFFALO       NY  -8.05  +3.83
1 GILROY        CA  +2.75  +1.97
9 COUNT=0010  TOTAL1=+13.55  TOTAL2=-02.90
```

Date Field Arithmetic

Introduction

You can use the BUILD or OVERLAY operands of the INREC, OUTREC and OUTFIL statements to perform various types of arithmetic on date fields in either Julian or Gregorian form, with 2-digit or 4-digit years, in CH, ZD and PD format.

Date field arithmetic uses new keywords as well as the following elements previously available for date field conversions:

- date: p,m,Y2x for a 2-digit year CH, ZD or PD input date field, or p,m,Y4x for a 4-digit year CH, ZD or PD input date field.
- todate: TOJUL to convert the resulting date to a Julian date, or TOGREG to convert the resulting date to a Gregorian date.

The new keywords and associated functions for date field arithmetic are as follows:

- **ADDDAYS, ADDMONS or ADDYEARS** can be used to add days, months or years to a date field. As a simple example, you could use the following to add 10 days to a C'ccyymmdd' date and produce the resulting C'ccyy/mm/dd' date:

  ```
  15,8,Y4T,ADDDAYS,+10,TOGREG=Y4T(/)
  ```

- **SUBDAYS, SUBMONS or SUBYEARS** can be used to subtract days, months or years from a date field. As a simple example, you could use the following to subtract 3 months from a P'dddyy' date and produce the resulting P'ccyyddd' date:

  ```
  21,3,Y2X,SUBMONS,+3,TOJUL=Y4U
  ```

- **DATEDIFF** can be used to calculate the number of days between two date fields. As a simple example, you could use the following to calculate the difference in days between two C'ccyymmdd' dates:

  ```
  41,8,Y4T,DATEDIFF,31,8,Y4T
  ```

- **NEXTDday** can be used to calculate the next specified day for a date field. As a simple example, you could use the following to calculate the next Friday for a C'ccyyddd' date as a C'ccyy.ddd' date:

  ```
  3,7,Y4T,NEXTDFRI,TOJUL=Y4T(.)
  ```

- **PREVDday** can be used to calculate the previous specified day for a date field. As a simple example, you could use the following to calculate the previous Wednesday for a P'yyddd' date as a C'ccyymmdd' date:

  ```
  51,3,Y2U,PREVDWED,TOGREG=Y4T
  ```

- **LASTDAYW, LASTDAYM, LASTDAYQ or LASTDAYY** can be used to calculate the last day of the week, month, quarter or year for a date field. As a simple example, you could use the following to calculate the last day of the month for a C'mmddccyy' date as a C'mmddccyy' date:

  ```
  28,8,Y4W,LASTDAYM,TOGREG=Y4W
  ```

DFSORT symbols can be used for fields and constants in date field arithmetic.

Syntax for Adding/Subtracting Days, Months or Years

```
p,m,Yxx,keyword,numeric,todate
```

Detailed Description for Adding/Subtracting Days, Months or Years

Can be used to add n days, months or years to a date, or subtract n days, months or years from a date. The valid keywords are ADDDAYS, ADDMONS, ADDYEARS, SUBDAYS, SUBMONS and SUBYEARS.

The numeric value (n) specifies the number of days, months or years to be added to or subtracted from the input date field. The resulting output date field is converted to the form indicated by todate.

- **p,m,Yxx** specifies the starting position (p), length (m) and format (Yxx) of a 2-digit or 4-digit year input date field. The valid length and format for each type of date field you can use is shown in Table 1.

  %nn (a parsed field) can be used for p,m.

Table 1. p,m,Yxx fields for date arithmetic

m,Yxx	**Type of date**
5,Y2T	C'yyddd' or Z'yyddd'
6,Y2T	C'yymmdd' or Z'yymmdd'
7,Y4T	C'ccyyddd' or Z'ccyyddd'
8,Y4T	C'ccyymmdd' or Z'ccyymmdd'
5,Y2W	C'dddyy' or Z'dddyy'
6,Y2W	C'mmddyy' or Z'mmddyy'
7,Y4W	C'dddccyy' or Z'dddccyy'
8,Y4W	C'mmddccyy' or Z'mmddccyy'
3,Y2U	P'yyddd'
4,Y2V	P'yymmdd'
4,Y4U	P'ccyyddd'
5,Y4V	P'ccyymmdd'
3,Y2X	P'dddyy'
4,Y2Y	P'mmddyy'
4,Y4X	P'dddccyy'
5,Y4Y	P'mmddccyy'

- **ADDDAYS** adds n days to the p,m,Yxx date field.
- **ADDMONS** adds n months to the p,m,Yxx date field. If necessary, the resulting date is adjusted backwards to the last valid day of the month. For example, if 1 month is added to 20101031, the resulting date will be 20101130 rather than 20101131.
- **ADDYEARS** adds n years to the p,m,Yxx date field. If the resulting date is February 29 of a non-leap year, it will be adjusted backwards to a valid date of February 28.
- **SUBDAYS** subtracts n days from the p,m,Yxx date field.
- **SUBMONS** subtracts n months from the p,m,Yxx date field. If necessary, the resulting date is adjusted backwards to the last valid day of the month. For example, if 1 month is subtracted from 20101031, the resulting date will be 20100930 rather than 20100931.

- **SUBYEARS** subtracts n years from the p,m,Yxx date field. If the resulting date is February 29 of a non-leap year, it will be adjusted backwards to a valid date of February 28.
- **numeric** can be a numeric constant (+n or -n), or a valid numeric field (p,m,f) with BI, FI, ZD, PD, FS, UFF or SFF format and an appropriately corresponding length. Refer to *z/OS DFSORT Application Programming Guide (SC26-7523-05)* for complete details on the numeric constants and numeric fields you can use.

  %nn (a parsed field) **cannot** be used for p,m.

  For ADDDAYS or SUBDAYS, the numeric constant or field value can be from -3652058 to +3652058.

  For ADDMONS or SUBMONS, the numeric constant or field value can be from -119987 to +119987.

  For ADDYEARS or SUBYEARS, the numeric constant or field value can be from -9998 to +9998.
- **todate** can be one of the following:
  - **TOJUL=Yaa**

    Converts the resulting date to a Julian output date.
  - **TOJUL=Yaa(s)**

    Converts the resulting date to a Julian output date with s separators. s can be any character except a blank.
  - **TOGREG=Yaa**

    Converts the resulting date to a Gregorian output date.
  - **TOGREG=Yaa(s)**

    Converts the resulting date to a Gregorian output date with s separators. s can be any character except a blank.

  The output date field created by each valid todate combination is shown in Table 2.

Table 2. TOJUL and TOGREG output date fields

Yaa	**TOJUL=Yaa**	**TOJUL=Yaa(s)**	**TOGREG=Yaa**	**TOGREG=Yaa(s)**
Y2T	C'yyddd'	C'yysddd'	C'yymmdd'	C'yysmmsdd'
Y2W	C'dddyy'	C'dddsyy'	C'mmddyy'	C'mmsddsyy'
Y2U	P'yyddd'	n/a	n/a	n/a
Y2V	n/a	n/a	P'yymmdd'	n/a
Y2X	P'dddyy'	n/a	n/a	n/a
Y2Y	n/a	n/a	P'mmddyy'	n/a
Y4T	C'ccyyddd'	C'ccyysddd'	C'ccyymmdd'	C'ccyysmmsdd'
Y4W	C'dddccyy'	C'dddsccyy'	C'mmddccyy'	C'mmsddsccyy'
Y4U	P'ccyyddd'	n/a	n/a	n/a
Y4V	n/a	n/a	P'ccyymmdd'	n/a
Y4X	P'dddccyy'	n/a	n/a	n/a
Y4Y	n/a	n/a	P'mmddccyy'	n/a

Syntax for Calculating Difference in Days

```
p,m,Yxx,DATEDIFF,p,m,Yxx
```

Detailed Description for Calculating Difference in Days

Can be used to calculate the number of days difference between two dates. The result is an 8-byte value consisting of a sign and 7 digits (sddddddd). If the first date is greater than or equal to the second date, the sign is + (plus). If the first date is less than the second date, the sign is - (minus).

- **p,m,Yxx** specifies the starting position (p), length (m) and format (Yxx) of each 2-digit or 4-digit year input date field. The valid length and format for each type of date field you can use is shown in Table 1 on page 17.

  %nn (a parsed field) can be used for p,m on the left side of DATEDIFF.

  %nn (a parsed field) **cannot** be used for p,m on the right side of DATEDIFF.

- **DATEDIFF** calculates the number of days that result when the second date is subtracted from the first date. The result is in the form sddddddd with + or - for s as appropriate and leading zeros. The result can be between -3652058 and +3652058 days.

Syntax for Next, Previous or Last Day

```
p,m,Yxx,keyword,todate
```

Detailed Description for Next, Previous or Last Day

Can be used to specify the next specified day, previous specified day, or last day of the week, month quarter or year for a date. The valid keywords are NEXTDday (where day can be SUN, MON, TUE, WED, THU, FRI or SAT), PREVDday (where day can be SUN, MON, TUE, WED, THU, FRI or SAT), LASTDAYW, LASTDAYM, LASTDAYQ or LASTDAYY.

The result is an output date field which is converted to the form indicated by todate.

- **p,m,Yxx** specifies the starting position (p), length (m) and format (Yxx) of the 2-digit or 4-digit year input date field. The valid length and format for each type of date field you can use is shown in Table 1 on page 17.

  %nn (a parsed field) can be used for p,m.

- **NEXTDSUN** calculates the next Sunday for a date field.
- **NEXTDMON** calculates the next Monday for a date field.
- **NEXTDTUE** calculates the next Tuesday for a date field.
- **NEXTDWED** calculates the next Wednesday for a date field.
- **NEXTDTHU** calculates the next Thursday for a date field.
- **NEXTDFRI** calculates the next Friday for a date field.
- **NEXTDSAT** calculates the next Saturday for a date field.
- **PREVDSUN** calculates the previous Sunday for a date field.
- **PREVDMON** calculates the previous Monday for a date field.
- **PREVDTUE** calculates the previous Tuesday for a date field.
- **PREVDWED** calculates the previous Wednesday for a date field
- **PREVDTHU** calculates the previous Thursday for a date field.
- **PREVDFRI** calculates the previous Friday for a date field.
- **PREVDSAT** calculates the previous Saturday for a date

- **LASTDAYW** calculates the last Friday of the week for a date field.
- **LASTDAYM** calculates the last day of the month for a date field.
- **LASTDAYQ** calculates the last day of the quarter for a date field.
- **LASTDAYY** calculates the last day of the year for a date field.
- **todate** converts the resulting date to a Julian or Gregorian output date. See the previous description of todate for details.

Arithmetic with Real Dates, Special Indicators and Invalid Dates

For CH/ZD dates (Y2T, Y4T, Y2W, Y4W), the zone and sign are ignored; only the digits are used. For example, X'F2F0F0F9F1F2F3D0' and X'A2B0C0D9E1F21320' are treated as 20091230. For PD dates (Y2U, Y4U, Y2V, Y4V, Y2X, Y4X, Y2Y, Y4Y), the sign is ignored; only the relevant digits are used. For example, X'120091230D' is treated as 20091230.

For CH/ZD dates (Y2T, Y4T, Y2W, Y4W), the special indicators are X'00...00' (BI zeros), X'40...40' (blanks), C'0...0' (CH zeros), Z'0...0' (ZD zeros), C'9...9' (CH nines), Z'9...9' (ZD nines) and X'FF...FF' (BI ones). For PD dates (Y2U, Y4U, Y2V, Y4V, Y2X, Y4X, Y2Y, Y4Y), the special indicators are P'0...0' (PD zeros) and P'9...9' (PD nines).

yy for real 2-digit year dates is transformed to ccyy when appropriate using the century window established by the Y2PAST option in effect. ccyy for real 4-digit year dates is transformed to yy when appropriate by removing cc.

If a special indicator is used for DATEDIFF, the result will be an output value of asterisks and an informational message (issued once).

Date arithmetic is not performed for special indicators; the special indicator is just used appropriately for the output date field. For example, if p,5,Y2T,ADDDAYS,+10,TOGREG=Y4T(/) is used, an input yyddd special indicator of C'99999' results in an output date field of C'9999/99/99'. However, CH/ZD special indicators of BI zeros, blanks and BI ones cannot be converted to PD special indicators.

Arithmetic involving an input date with an invalid digit (A-F) will result in a data exception (0C7 ABEND) or an incorrect output value.

Arithmetic involving an invalid input date or invalid output date will result in an output value of asterisks and an informational message (issued once). A date is considered invalid if any of the following range conditions are not met:

- yy must be between 00 and 99
- ccyy must be between 0001 and 9999
- mm must be between 01 and 12
- dd must be between 01 and 31, and must be valid for the year and month
- ddd must be 001 to 366 for a leap year, or between 001 and 365 for a non-leap year.

A date is also considered invalid if the input field is a CH/ZD special indicator of binary zeros, blanks or binary ones, and the output field is PD.

Example 1

```
  OPTION COPY
  INREC OVERLAY=(11:1,8,Y4T,ADDDAYS,+50,TOGREG=Y4T,
              21:1,8,Y4T,SUBMONS,+7,TOGREG=Y4T,
              31:1,8,Y4T,ADDYEARS,+2,TOGREG=Y4T)
  OUTFIL REMOVECC,
   HEADER1=('Input     +50 days  -7 months +2 years')
```

This example illustrates how you can add and subtract days, months and years from date fields.

The SORTIN data set has these input records with a C'ccyymmdd' date field in positions 1-8:

```
20070305
20071213
20080219
20080901
20091122
20090115
20100915
20100630
99999999
```

The INREC statement performs three different date field arithmetic operations on the input date field. It adds 50 days, subtracts 7 months and adds 2 years. We use 1,8,Y4T for the input field to match the C'ccyymmdd' input date, and we use TOGREG=Y4T to give us a C'ccyymmdd' output date. The OUTFIL statement adds headings.

SORTOUT will have these records:

```
Input     +50 days  -7 months +2 years
20070305  20070424  20060805  20090305
20071213  20080201  20070513  20091213
20080219  20080409  20070719  20100219
20080901  20081021  20080201  20100901
20091122  20100111  20090422  20111122
20090115  20090306  20080615  20110115
20100915  20101104  20100215  20120915
20100630  20100819  20091130  20120630
99999999  99999999  99999999  99999999
```

Note that the '99999999' input value is treated as a special indicator for output.

Example 2

```
  OPTION COPY,Y2PAST=1990
  OUTREC IFTHEN=(WHEN=INIT,
    BUILD=(1:1,8,UFF,TO=ZD,LENGTH=6,8:11,6,UFF,TO=ZD,LENGTH=5)),
   IFTHEN=(WHEN=INIT,
     BUILD=(1,6,Y2W,DATEDIFF,8,5,Y2T))
```

This example illustrates how you can calculate the difference in days between two different types of date fields, each of which has separators.

The SORTIN data set has these input records with a C'mm/dd/yy' date field in positions 1-8 and a C'yy/ddd' date field in positions 11-16:

```
03/05/07 07/052
12/13/07 08/193
02/19/08 08/365
09/01/08 09/001
11/22/09 09/015
07/22/98 01/121
01/15/09 09/015
09/15/10 09/322
06/30/10 08/050
```

The OUTREC statement calculates the number of days for date1-date2 and puts the result in the output record in positions 1-8.

The first IFTHEN clause removes the / separators from date1 and date2 so we can use them in DATEDIFF. After the first IFTHEN clause, the records look like this, with the C'mmddyy' date in positions 1-5 and the C'yyddd' date in positions 8-12:

```
030507 07052
121307 08193
021908 08365
090108 09001
112209 09015
072298 01121
011509 09015
091510 09322
063010 08050
```

The second IFTHEN clause uses DATEDIFF to get the number of days between date1 and date2. We use 1,6,Y2W to match the C'mmddyy' date and 8,5,Y2T to match the C'yyddd' date.

SORTOUT will have these records:

```
+0000012
-0000211
-0000315
-0000122
+0000311
-0001014
+0000000
+0000301
+0000862
```

Note that when date1>=date2, the result is a positive value, and when date1<date2, the result is a negative value.

Example 3

```
  OPTION Y2PAST=1990
  SORT FIELDS=(1,6,Y2W,A)
  OUTFIL REMOVECC,
     HEADER1=(1:'Input',15:'NEXTDFRI',25:'PREVDSUN',35:'LASTDAYQ'),
     BUILD=(1:1,6,Y2W,TOJUL=Y4W(-),
         15:1,6,Y2W,NEXTDFRI,TOJUL=Y4W(-),
         25:1,6,Y2W,PREVDSUN,TOJUL=Y4W(-),
         35:1,6,Y2W,LASTDAYQ,TOJUL=Y4W(-))
```

This example illustrates how you can sort by a date, and calculate a specific day after and before a date, and the last day of the quarter for a date. The input date is in the form C'mmddyy' and the output dates will be in the form 'ddd-yyyy'.

The SORTIN data set has these input records with a C'mmddyy' date field in positions 1-6:

```
010105
120699
021610
999999
092810
031500
000000
032505
110210
```

The SORT statement sorts by the C'mmddyy' date. We use 1,6,Y2W for the sort field to match the C'mmddyy' date.

The OUTFIL statement writes a heading and performs four operations on each date. It converts the input date to C'ddd-yyyy' form, gets the next Friday date in C'ddd-yyyy' form, gets the previous Sunday date in C'ddd-yyyy' form, and gets the end of the quarter date in C'ddd-yyyy' form. We use 1,6,Y2W for the input field to match the C'mmddyy' date, and we use TOJUL=Y4W(-) to give us a C'ddd-yyyy' output date.

SORTOUT will have these records:

```
Input         NEXTDFRI  PREVDSUN  LASTDAYQ
000-0000      000-0000  000-0000  000-0000
340-1999      344-1999  339-1999  365-1999
075-2000      077-2000  072-2000  091-2000
001-2005      007-2005  361-2004  090-2005
084-2005      091-2005  079-2005  090-2005
047-2010      050-2010  045-2010  090-2010
271-2010      274-2010  269-2010  273-2010
306-2010      309-2010  304-2010  365-2010
999-9999      999-9999  999-9999  999-9999
```

Note that the '000000' and '999999' input values are treated as special indicators for output.

SET and PROC symbols in control statements

Introduction

For jobs that directly invoke DFSORT (PGM=SORT or PGM=ICEMAN) or ICETOOL (PGM=ICETOOL), you can construct DFSORT symbols that incorporate JCL PROC or SET symbols as well as text and system symbols. You can then use those constructed DFSORT symbols in DFSORT and ICETOOL control statements in the same way you use regular DFSORT symbols.

For complete information on DFSORT Symbols, see *z/OS DFSORT Application Programming Guide (SC26-7523-05)*.

As a simple example, if you wanted to use SET symbols called XDSN and YDSN in a DFSORT statement, you could code the following:

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

For each JPn"string" parameter found in EXEC PARM, a DFSORT symbol in the following form is constructed, and treated as if it was specified in the SYMNAMES data set (even if a SYMNAMES data set is not actually present):

```
JPn,S'string'
```

Up to ten symbols, JP0-JP9, can be set up this way.

For the example above, the following DFSORT symbols are constructed and treated as if they were specified in SYMNAMES:

```
JP1,S'X.DATA'
JP2,S'Y.DATA'
```

These symbols are placed in the Symbol Table as:

```
JP1,C'X.DATA'
JP2,C'Y.DATA'
```

When JP1 and JP2 are used in the OMIT statement, the statement is transformed to:

```
 OMIT COND=(1,44,CH,EQ,C'X.DATA',OR,1,44,CH,EQ,C'Y.DATA')
```

Here's another example using ICETOOL:

```
// SET VLSR1='339001'
// SET VLSR2='339002'
//S2 EXEC PGM=ICETOOL,
// PARM='JP0"Report for disks &VLSR1 and &VLSR2 on &WDAY"'
...
//TOOLIN DD *
DISPLAY TITLE(JP0) -
  FROM(IN) LIST(RPT) ON(5,4,CH)
```

The following DFSORT symbol is constructed:

```
JP0,S'Report for disks 339001 and 339002 on &WDAY'
```

If the job is run on Friday, 'FRI' is substituted for the &WDAY system symbol and the following is placed in the Symbol Table:

```
JP0,C'Report for disks 339001 and 339002 on FRI'
```

When JP0 is used in the DISPLAY statement, the statement is transformed to:

```
DISPLAY TITLE('Report for disks 339001 and 339002 on FRI')-
FROM(IN) LIST(OUT) ON(1,5,CH)
```

Syntax

- For DFSORT:

  ```
  //stepname EXEC PGM=SORT,PARM='option<,option>...'
  ```

option can be JPn"string" or one of the other valid DFSORT options (for example, VLSCMP or DSA=8). JPn"string" options are used to construct DFSORT symbols. Other DFSORT options are passed to DFSORT as EXEC PARM options in the normal way.

PGM=ICEMAN or one of the other aliases can be used instead of PGM=SORT.

Normal system JCL rules for the EXEC PARM operand apply.

Notes:

1. JPn"string" parameters can only be used when DFSORT is invoked directly (for example, with PGM=SORT), not when DFSORT is invoked from a program (for example, with LINK EP=SORT).
2. If the length of the EXEC PARM options is greater than the JCL limit of 100 bytes, JPn"string" options will not be processed.

- For ICETOOL:

  ```
  //stepname EXEC PGM=ICETOOL,PARM='option<,option>...'
  ```

  option can be JPn"string". JPn"string" options are used to construct DFSORT symbols. Other options are ignored by ICETOOL.

  Normal system JCL rules for the EXEC PARM operand apply.

  Notes:

  1. JPn"string" parameters can only be used when ICETOOL is invoked directly (for example, with PGM=ICETOOL), not when ICETOOL is invoked from a program (for example, with LINK EP=ICETOOL).
  2. If the length of the EXEC PARM options is greater than the JCL limit of 100 bytes, JPn"string" options will not be processed.

Detailed Description

JPn"string"

Only JP0 through JP9 can be used for a symbol name where 'JP' must be uppercase EBCDIC (X'D1D7') and n must be '0'-'9' (X'F0'-X'F9').

A quote must appear before and after the string. A quote must not appear within the string (it would be interpreted as the ending quote). PARM='...,JPn"string' (a missing ending quote) will be interpreted as PARM='...,JPn"string"'.

If the keyword does not appear as JPn"string", it will not be used as a JPn symbol. For example, JP1"ABC" would be used as a JP1 symbol, but JP1="ABC", JP1("ABC"), JP1=("ABC") or jp1"ABC" would not.

Any or all of the following can appear in the string:

- any EBCDIC character (except quote). If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes
- a JCL SET or PROC symbol (&SYMBOL). Normal system rules for SET and PROC symbols apply.
- a system symbol (for example, &JOBNAME, &SYSPLEX, etc). Normal system rules for system symbols apply.

The symbol will be constructed as:

```
JPn,S'string'
```

and must conform to the rules for a system symbol string as documented in *z/OS DFSORT Application Programming Guide (SC26-7523-05)*.

If you specify:

```
//SYMNOUT DD SYSOUT=*
```

the JPn symbols will be listed along with any other DFSORT symbols you specify in SYMNAMES.

If you specify a SYMNAMES data set, the JPn symbols will be processed **before** the first SYMNAMES symbol.

Note: Here's an example of one suggested way of specifying multiple options in EXEC PARM:

```
//  SET X1='ANY-STRING'
//  SET X2='ANOTHER-STRING'
//S1 EXEC PGM=SORT,
//  PARM=('RESALL=12K',
//   'VLLONG',
//   'JP1"&X1"',
//   'JP2"&X2"')
```

Example 1

MYPROC Procedure

```
//MYPROC PROC DAY=
//STEP01 EXEC PGM=SORT,PARM='JP1"&DAY"'
//SYSOUT  DD SYSOUT=*
//SYMNOUT DD SYSOUT=*
//SORTIN DD DSN=MYCNTL.PDS(SEL),DISP=SHR
//SORTOUT DD SYSOUT=*
//SYSIN DD DSN=MYCNTL.PDS(OVLY),DISP=SHR
//  PEND
```

Execution of MYPROC

```
//S1 EXEC MYPROC,DAY=2
```

Record in SEL member

```
SELECT * FROM MYTABLE WHERE TM_RECEIPT >= (CURRENT DATE - ? DAYS)
```

Records in OVLY member

```
  OPTION COPY
  INREC OVERLAY=(59:JP1)
```

This example illustrates how you can use a JCL PROC symbol in a DFSORT control statement.

The JCL PROC symbol is DAY which will be set to a 1 character numeric value (2 in this example), and used to overlay the number of days in the SELECT statement (in the SEL member).

PARM='JP1"&DAY"' is used to create a DFSORT symbol named JP1 containing a string with the DAY value.

The SYMNOUT listing will show the following:

```
------ SYMNAMES STATEMENTS FROM EXEC PARM -------
JP1,S'2'

------------------ SYMBOL TABLE -----------------
JP1,C'2'
```

The DFSORT INREC statement in the OVLY member uses JP1. It will be transformed to:

```
  INREC OVERLAY=(59:C'2')
```

and used to change the SELECT statement to the following in SORTOUT:

```
SELECT * FROM MYTABLE WHERE TM_RECEIPT >= (CURRENT DATE - 2 DAYS)
```

Example 2

```
//  SET JOBNM='FRANK2'
//S1    EXEC  PGM=ICETOOL,
//  PARM=('JP1"&JOBNM"',
//        'JP2"&JOBNM STEP"',
//        'JP3"&JOBNM EXCPS"')
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//SYMNAMES DD *
JOBN,5,8,CH
STEPN,*,8,CH
EXCPS,*,5,ZD
//SYMNOUT DD SYSOUT=*
//IN DD *
    FRANK2  S1      00008
    FRANK2  S2      00123
    FRANK2  S3      00023
    FRANK3  S1      00016
    FRANK3  S2      00152
/*
//T1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//RPT DD SYSOUT=*
//TOOLIN DD *
COPY FROM(IN) TO(T1) USING(CTL1)
DISPLAY FROM(T1) LIST(RPT) BLANK -
 HEADER(JP2) ON(STEPN) -
 HEADER(JP3) ON(EXCPS)
/*
//CTL1CNTL DD *
  INCLUDE COND=(JOBN,EQ,JP1)
/*
```

This example illustrates how you can use a JCL SET symbol in ICETOOL and DFSORT control statements.

The JCL SET symbol is JOBNM which is set to a constant ('FRANK2' for this example).

```
//  PARM=('JP1"&JOBNM"',
//        'JP2"&JOBNM STEP"',
//        'JP3"&JOBNM EXCPS"')
```

is used to create DFSORT symbols named JP1, JP2 and JP3 containing strings including the JOBNM value. ICETOOL treats these JPn symbols as if they were specifed before the Symbols in SYMNAMES.

The SYMNOUT listing will show the following:

```
------ SYMNAMES STATEMENTS FROM EXEC PARM -------
JP1,S'FRANK2'
JP2,S'FRANK2 STEP'
JP3,S'FRANK2 EXCPS'

------- ORIGINAL STATEMENTS FROM SYMNAMES -------
JOBN,5,8,CH
STEPN,*,8,CH
EXCPS,*,5,ZD

------------------ SYMBOL TABLE -----------------
JP1,C'FRANK2'
JP2,C'FRANK2 STEP'
JP3,C'FRANK2 EXCPS'
JOBN,5,8,CH
STEPN,13,8,CH
EXCPS,21,5,ZD
```

The DFSORT INCLUDE statement in CTL1CNTL uses the JOBN and JP1 symbols. It will be transformed to:

```
 INCLUDE COND=(5,8,CH,EQ,C'FRANK2')
```

The ICETOOL DISPLAY statement uses the JP2, JP3, STEPN and EXCPS symbols. It will be transformed to:

```
DISPLAY FROM(T1) LIST(RPT) BLANK-
HEADER('FRANK2 STEP') ON(13,8,CH)-
HEADER('FRANK2 EXCPS') ON(21,5,ZD)
```

RPT will contain the following report:

```
FRANK2 STEP   FRANK2 EXCPS
-----------   ------------
S1                       8
S2                     123
S3                      23
```

Accept n records with OUTFIL

Description

ACCEPT=n is a new option for the existing OUTFIL statement. ACCEPT=n specifies the number of OUTFIL input records to be accepted for processing for this OUTFIL group. A record is accepted if it is not deleted by STARTREC, ENDREC, SAMPLE, INCLUDE or OMIT processing for the group. After n OUTFIL input records have been accepted, additional OUTFIL input records are not included in the data sets for this OUTFIL group.

n specifies the number of records to be accepted. The value for n starts at 1 and is limited to 28 digits (15 significant digits).

If ACCEPT=n is specified with SAVE, it will be applied to the saved records.

ACCEPT=n operates in a similar way to the ENDREC=n option. However, whereas ENDREC=n stops processing OUTFIL input records for a group at relative record n, ACCEPT=n stops processing OUTFIL input records for a group after n records have been "accepted". If both ACCEPT=n and ENDREC=n are specified, processing will stop when the first criteria is met.

Example 1

```
  OPTION COPY
  OUTFIL FNAMES=OUT1,INCLUDE=(11,3,CH,EQ,C'D51'),ACCEPT=3
  OUTFIL FNAMES=(OUT2A,OUT2B),STARTREC=2,ACCEPT=5
  OUTFIL FNAMES=OUT3,INCLUDE=(11,3,CH,EQ,C'D51'),ACCEPT=3,ENDREC=5
```

This example illustrates how you can stop processing OUTFIL input records for various groups in different ways.

The SORTIN data set has these input records:

```
HEADER  2010/06/30
FRANK     D51
ED        D52
VICKY     D51
MARTIN    D52
LILY      D50
MARC      D51
JUNE      D51
LUCY      D51
TRAILER     8
```

The first OUTFIL statement uses INCLUDE to extract the D51 records and ACCEPT=3 to only write the first three D51 records to OUT1. Thus, OUT1 will have these records:

```
FRANK     D51
VICKY     D51
MARC      D51
```

The second OUTFIL statement uses STARTREC=2 to skip the HEADER record and ACCEPT=5 to only write the next five records to OUT2A and OUT2B. Thus, OUT2A and OUT2B will have these records:

```
FRANK     D51
ED        D52
VICKY     D51
MARTIN    D52
LILY      D50
```

The third OUTFIL statement uses INCLUDE to extract the D51 records, ENDREC=5 to stop at relative record 5 (MARTIN), and ACCEPT=3 to accept a maximum of three records. In this case, there are only two D51 records before relative record 5 so ENDREC=5 is satisfied before ACCEPT=3 and only two records are written to OUT3 as follows:

```
FRANK     D51
VICKY     D51
```

Example 2

```
  OPTION COPY
  OUTFIL FNAMES=X1,STARTREC=2,ENDREC=5
  OUTFIL FNAMES=X2,SAVE,ACCEPT=3
```

This example illustrates how you can use ACCEPT=n with SAVE processing.

The SORTIN data set has these input records:

```
R01
R02
R03
R04
R05
R06
R07
R08
R09
R10
```

The first OUTFIL statement uses STARTREC=2 and ENDREC=5 to write relative records two through five to X1. Thus, X1 will have these records:

```
R02
R03
R04
R05
```

The second OUTFIL statement uses SAVE and ACCEPT=3, so it will write the first three saved records to X2. The saved records are those NOT written to X1 (R01, R06-R10), so X2 will have these records:

```
R01
R06
R07
```

New translation functions

Description

The functions available with DFSORT's p,m,TRAN=keyword and p,TRAN=keyword operands have been expanded. p,m,TRAN=keyword translates a specified field where p is the starting position and m is the length. p,TRAN=keyword translates the variable part of the record. See *z/OS DFSORT Application Programming Guide (SC26-7523-05)* for details of using p,m,TRAN=keyword and p,TRAN=keyword.

%nn (a parsed field) can be used for p,m.

A DFSORT symbol can be used for p,m or p.

The following new keywords are now supported for p,m,TRAN=keyword and p,TRAN=keyword, in addition to the existing UTOL, LTOU and ALTSEQ keywords:

- **p,m,TRAN=ATOE and p,TRAN=ATOE**

  Translates ASCII characters to their equivalent EBCDIC characters using the default standard TCP/IP service ASCII-to-EBCDIC table shown below, where each column shows the ASCII (A) character value on the left and its equivalent EBCDIC (E) character value on the right.

  m can be 1 to 32752.

TRAN=ATOE ASCII-to-EBCDIC Table - Part 1

A	E	A	E	A	E	A	E	A	E	A	E	A	E	A	E
00	00	10	10	20	40	30	F0	40	7C	50	D7	60	79	70	97
01	01	11	11	21	5A	31	F1	41	C1	51	D8	61	81	71	98
02	02	12	12	22	7F	32	F2	42	C2	52	D9	62	82	72	99
03	03	13	13	23	7B	33	F3	43	C3	53	E2	63	83	73	A2
04	37	14	3C	24	5B	34	F4	44	C4	54	E3	64	84	74	A3
05	2D	15	3D	25	6C	35	F5	45	C5	55	E4	65	85	75	A4
06	2E	16	32	26	50	36	F6	46	C6	56	E5	66	86	76	A5
07	2F	17	26	27	7D	37	F7	47	C7	57	E6	67	87	77	A6
08	16	18	18	28	4D	38	F8	48	C8	58	E7	68	88	78	A7
09	05	19	19	29	5D	39	F9	49	C9	59	E8	69	89	79	A8
0A	25	1A	3F	2A	5C	3A	7A	4A	D1	5A	E9	6A	91	7A	A9
0B	0B	1B	27	2B	4E	3B	5E	4B	D2	5B	AD	6B	92	7B	C0
0C	0C	1C	22	2C	6B	3C	4C	4C	D3	5C	E0	6C	93	7C	4F
0D	0D	1D	1D	2D	60	3D	7E	4D	D4	5D	BD	6D	94	7D	D0
0E	0E	1E	35	2E	4B	3E	6E	4E	D5	5E	5F	6E	95	7E	A1
0F	0F	1F	1F	2F	61	3F	6F	4F	D6	5F	6D	6F	96	7F	07

TRAN=ATOE ASCII-to-EBCDIC Table - Part 2

A	E	A	E	A	E	A	E	A	E	A	E	A	E	A	E
80	00	90	10	A0	40	B0	F0	C0	7C	D0	D7	E0	79	F0	97
81	01	91	11	A1	5A	B1	F1	C1	C1	D1	D8	E1	81	F1	98
82	02	92	12	A2	7F	B2	F2	C2	C2	D2	D9	E2	82	F2	99
83	03	93	13	A3	7B	B3	F3	C3	C3	D3	E2	E3	83	F3	A2
84	37	94	3C	A4	5B	B4	F4	C4	C4	D4	E3	E4	84	F4	A3
85	2D	95	3D	A5	6C	B5	F5	C5	C5	D5	E4	E5	85	F5	A4
86	2E	96	32	A6	50	B6	F6	C6	C6	D6	E5	E6	86	F6	A5
87	2F	97	26	A7	7D	B7	F7	C7	C7	D7	E6	E7	87	F7	A6
88	16	98	18	A8	4D	B8	F8	C8	C8	D8	E7	E8	88	F8	A7
89	05	99	19	A9	5D	B9	F9	C9	C9	D9	E8	E9	89	F9	A8
8A	25	9A	3F	AA	5C	BA	7A	CA	D1	DA	E9	EA	91	FA	A9
8B	0B	9B	27	AB	4E	BB	5E	CB	D2	DB	AD	EB	92	FB	C0
8C	0C	9C	22	AC	6B	BC	4C	CC	D3	DC	E0	EC	93	FC	4F
8D	0D	9D	1D	AD	60	BD	7E	CD	D4	DD	BD	ED	94	FD	D0
8E	0E	9E	35	AE	4B	BE	6E	CE	D5	DE	5F	EE	95	FE	A1
8F	0F	9F	1F	AF	61	BF	6F	CF	D6	DF	6D	EF	96	FF	07

As an example, TRAN=ATOE would translate ASCII aB2 (X'614232') to EBCDIC aB2 (X'81C2F2').

- **p,m,TRAN=ETOA and p,TRAN=ETOA**

  Translates EBCDIC characters to their equivalent ASCII characters using the default standard TCP/IP service EBCDIC-to-ASCII table shown below, where each column shows the EBCDIC (E) character value on the left and its equivalent ASCII (A) character value on the right.

  m can be 1 to 32752.

TRAN=ETOA EBCDIC-to-ASCII Table - Part 1

E	A	E	A	E	A	E	A	E	A	E	A	E	A	E	A
00	00	10	10	20	1A	30	1A	40	20	50	26	60	2D	70	D7
01	01	11	11	21	1A	31	1A	41	A6	51	A9	61	2F	71	88
02	02	12	12	22	1C	32	16	42	E1	52	AA	62	DF	72	94
03	03	13	13	23	1A	33	1A	43	80	53	9C	63	DC	73	B0
04	1A	14	1A	24	1A	34	1A	44	EB	54	DB	64	9A	74	B1
05	09	15	0A	25	0A	35	1E	45	90	55	A5	65	DD	75	B2
06	1A	16	08	26	17	36	1A	46	9F	56	99	66	DE	76	FC
07	7F	17	1A	27	1B	37	04	47	E2	57	E3	67	98	77	D6
08	1A	18	18	28	1A	38	1A	48	AB	58	A8	68	9D	78	FB
09	1A	19	19	29	1A	39	1A	49	8B	59	9E	69	AC	79	60
0A	1A	1A	1A	2A	1A	3A	1A	4A	9B	5A	21	6A	BA	7A	3A
0B	0B	1B	1A	2B	1A	3B	1A	4B	2E	5B	24	6B	2C	7B	23
0C	0C	1C	1C	2C	1A	3C	14	4C	3C	5C	2A	6C	25	7C	40
0D	0D	1D	1D	2D	05	3D	15	4D	28	5D	29	6D	5F	7D	27
0E	0E	1E	1E	2E	06	3E	1A	4E	2B	5E	3B	6E	3E	7E	3D
0F	0F	1F	1F	2F	07	3F	1A	4F	7C	5F	5E	6F	3F	7F	22

TRAN=ETOA EBCDIC-to-ASCII Table - Part 2

E	A	E	A	E	A	E	A	E	A	E	A	E	A	E	A
80	F8	90	8C	A0	C8	B0	B5	C0	7B	D0	7D	E0	5C	F0	30
81	61	91	6A	A1	7E	B1	B6	C1	41	D1	4A	E1	E7	F1	31
82	62	92	6B	A2	73	B2	FD	C2	42	D2	4B	E2	53	F2	32
83	63	93	6C	A3	74	B3	B7	C3	43	D3	4C	E3	54	F3	33
84	64	94	6D	A4	75	B4	B8	C4	44	D4	4D	E4	55	F4	34
85	65	95	6E	A5	76	B5	B9	C5	45	D5	4E	E5	56	F5	35
86	66	96	6F	A6	77	B6	E6	C6	46	D6	4F	E6	57	F6	36
87	67	97	70	A7	78	B7	BB	C7	47	D7	50	E7	58	F7	37
88	68	98	71	A8	79	B8	BC	C8	48	D8	51	E8	59	F8	38
89	69	99	72	A9	7A	B9	BD	C9	49	D9	52	E9	5A	F9	39
8A	96	9A	97	AA	EF	BA	8D	CA	CB	DA	A1	EA	A0	FA	B3
8B	A4	9B	87	AB	C0	BB	D9	CB	CA	DB	AD	EB	85	FB	F7
8C	F3	9C	CE	AC	DA	BC	BF	CC	BE	DC	F5	EC	8E	FC	F0
8D	AF	9D	93	AD	5B	BD	5D	CD	E8	DD	F4	ED	E9	FD	FA
8E	AE	9E	F1	AE	F2	BE	D8	CE	EC	DE	A3	EE	E4	FE	A7
8F	C5	9F	FE	AF	F9	BF	C4	CF	ED	DF	8F	EF	D1	FF	FF

As an example, TRAN=ETOA would translate EBCDIC aB2 (X'81C2F2') to ASCII aB2 (X'614232').

- **p,m,TRAN=HEX and p,TRAN=HEX**

  Translates binary values to their equivalent EBCDIC hexadecimal values. p,m,TRAN=HEX is equivalent in function to p,m,HEX. p,TRAN=HEX is equivalent in function to p,HEX.

  m can be 1 to 16376. The output length will be m*2.

  As an example, TRAN=HEX would translate X'C1F1' (C'A1') to C'C1F1'.

- **p,m,TRAN=UNHEX and p,TRAN=UNHEX**

  Translates EBCDIC hexadecimal values to their equivalent binary values.

  m can be 1 to 32752. The output length will be (m+1)/2.

  As an example, TRAN=UNHEX would translate C'C1F1' to X'C1F1' (C'A1').

  If m is odd, the last nibble will be 0. As an example, C'C1F1F' would translate to X'C1F1F0' (C'A10').

If an input character is not 0-9 or A-F, the equivalent nibble will be set to 0. As an example, C'G2' will be translated to X'02'.

- **p,m,TRAN=BIT and p,TRAN=BIT**

  Translates binary values to their equivalent EBCDIC bit values.

  m can be 1 to 4094. The output length will be m*8.

  As an example, TRAN=BIT would translate X'C1F1' (C'A1') to C'1100000111110001'

- **p,m,TRAN=UNBIT and p,TRAN=UNBIT**

  Translates EBCDIC bit values to their equivalent binary values.

  m can be 1 to 32752. The output length will be (m+7)/8.

  As an example, TRAN=UNBIT would translate C'1100000111110001' to X'C1F1' (C'A1').

  If m is not a multiple of 8, the missing bits on the right will be set to 0. As an example, C'1100000111111' would translate to X'C1F8'.

  If an input character is not 0 or 1, the equivalent bit will be set to 0. As an example, C'11100005' will be translated to X'E0'.

Example 1

```
  OPTION COPY
  OUTFIL FNAMES=OUT1,OVERLAY=(21:21,10,TRAN=ATOE)
  OUTFIL FNAMES=OUT2,BUILD=(1,80,TRAN=ATOE)
```

The input records will be copied to the OUT1 data set with the characters in positions 21-30 translated from ASCII to EBCDIC using the default standard TCP/IP service ASCII-to-EBCDIC table.

The input records will be copied to the OUT2 data set with the characters in positions 1-80 translated from ASCII to EBCDIC using the default standard TCP/IP service ASCII-to-EBCDIC table.

Example 2

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

The variable-length input records will be copied to the SORTOUT data set with the data from position 5 to the end of each record translated from EBCDIC to ASCII using the default standard TCP/IP service EBCDIC-to-ASCII table.

Example 3

```
  OPTION COPY
  INREC BUILD=(1,8,TRAN=UNHEX)
```

This example illustrates how you can translate EBCDIC hexadecimal values to their equivalent binary values.

The SORTIN data set has these 8-byte records:

```
C1F2E25B
E94DF05D
```

The SORTOUT data set has these 4-byte records:

```
A2S$
Z(0)
```

Begin group when key changes

Description

KEYBEGIN=(p,m) is a new option for the existing WHEN=GROUP function. It operates in a similar way to the BEGIN=(cond) option. However, whereas BEGIN=(cond) tells DFSORT to start a group when a condition is satisfied, KEYBEGIN=(p,m) tells DFSORT to start a group when the binary value in the specified key field (p,m) changes. Thus you can start a new group for each consecutive set of key values in your records.

p is the starting position of the key and can be from 1 to 32752.

m is the length of the key and can be from 1 to 256.

If KEYBEGIN=(p,m) is specified, the first input record will start a group.

A DFSORT symbol can be used for p,m.

Example

```
  SORT FIELDS=(1,12,CH,A,13,8,CH,D)
  OUTREC IFTHEN=(WHEN=GROUP,KEYBEGIN=(1,12),
    PUSH=(13:13,8,31:ID=3))
```

This example illustrates how you can propagate a value and group identifier to each group of records with the same key.

The SORTIN data set has these input records:

```
Bluejay     2010/003     26
Bluejay     2010/001     13
Bluejay     2010/015    152
Raven       2010/005      7
Raven       2010/025     14
Raven       2010/010     93
Finch       2010/090     21
Finch       2010/017      5
```

We SORT by the bird name ascending, and by the date descending, to get the highest date for each bird name. Then we use WHEN=GROUP with KEYBEGIN on the bird name to PUSH the highest date and a group identifier to each record of each group (bird name).

The SORTOUT data set has these output records:

```
Bluejay     2010/015    152   001
Bluejay     2010/015     26   001
Bluejay     2010/015     13   001
Finch       2010/090     21   002
Finch       2010/090      5   002
Raven       2010/025     14   003
Raven       2010/025     93   003
Raven       2010/025      7   003
```

Larger header/trailer fields for OUTFIL

Description

For a p,m field in a HEADER1, TRAILER1, HEADER2, TRAILER2, HEADER3 or TRAILER3 operand of an OUTFIL statement, m (the length of the input field in bytes) can now be up to 32752, removing the previous limit of 256. Note that the existing limits still apply for p,m,f fields.

RC8 and RC12 for COUNT

Description

RC8 can now be used to set RC=8 for a COUNT operator if the record count meets the specified criteria. RC8 can only be specified if EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v), or NOTEQUAL(w) is specified. RC8 overrides the default of setting RC=12 for a COUNT operator if the record count meets the specified criteria.

RC12 can now be used to set RC=12 for a COUNT operator if the record count meets the specified criteria. RC12 can only be specified if EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v), or NOTEQUAL(w) is specified. RC12 is equivalent to the default of not specifying RC4 or RC8.

Example

```
* Set RC=8 if INPUT1 is empty or
* set RC=0 if INPUT1 is not empty.
COUNT FROM(INPUT1) EMPTY RC8
```

More fields for DISPLAY

Description

Up to 50 HEADER and ON operands can now be used for a DISPLAY operator, removing the previous limit of 20 HEADER and ON operands.

Larger fields for ICETOOL

Description

For an ON(p,m,CH) operand in a DISPLAY, OCCUR, SELECT, SPLICE or UNIQUE operator, m (the length of the input field in bytes) can now be up to 4000, removing the previous limit of 1500.

For an ON(p,m,HEX) operand in a DISPLAY or OCCUR operator, m (the length of the input field in bytes) can now be up to 2000, removing the previous limit of 1000.

Longer reports for DISPLAY/OCCUR

Description

A DISPLAY or OCCUR report can now be up to 8192 bytes, removing the previous limit of 2048 bytes. For WIDTH(n), n can now be 121 to 8192. If WIDTH(n) is not specified, the calculated line length can be up to 8192 bytes if the new LONGLINE operand is specified, or up to 2048 bytes if LONGLINE is not specified. NOCC will continue to lower the line length limit by 1 byte.

Example

```
DISPLAY FROM(IN1) LIST(RPT1) LONGLINE BLANK -
  ON(1,4000,CH)
DISPLAY FROM(IN2) LIST(RPT2) BLANK -
  ON(1,5,PD) ON(21,18,CH) ON(21,18,HEX) WIDTH(3000)
```

Timestamp with microseconds

Description

A new DATE5 keyword can be used to create a run-time timestamp constant in the form 'yyyy-mm-dd-hh.mm.ss.nnnnnn' with the year (yyyy), month (mm), day (dd), hours (hh), minutes (mm), seconds (ss) and micro-seconds (nnnnnn). The DATE5 keyword can be used in logical conditions and for reformatting in the same way that the existing DATE4 keyword can be used. The constant created by the DATE5 keyword is the same as the constant created by the DATE4 keyword with the addition of '.nnnnnn' for microseconds at the end.

The DATE5 constant (like the DATE4 constant) reflects the timestamp at the beginning of the current run.

New Messages

This section shows messages that have been added for PTFs UK90025 and UK90026. Refer to *z/OS DFSORT Messages, Codes and Diagnosis Guide (SC26-7525-05)* for general information on DFSORT messages.

ICE265A

ICE265A TRLUPD FIELD ERROR

Explanation: Critical. The TRLUPD parameter of an OUTFIL statement contained an invalid column, keyword, position, length, format, pattern or sign. Some common errors are:

- A 0 value was used.
- A column was greater than 32752 or was followed by another column.
- A column overlapped the previous output field in the record.
- A keyword other than COUNT, COUNT+n, COUNT-n, TOTAL or TOT was specified.
- A position plus length for a TOTAL field was greater than 32753.
- The length for a TOTAL field was greater than 8 for BI or FI, 16 for PD, 31 for ZD, 32 for CSF/FS, or 44 for UFF or SFF.
- The length for a TOTAL field was not 4 or 8 for FL.
- More than 31 digits or 44 characters were specified in an edit pattern.

- SIGNz (where z is not S) was specified with Mn or without EDIT or EDxy.
- x, y, or z in EDxy or SIGNz were the same character.
- The value for LENGTH was greater than 44.

System Action: The program terminates.

Programmer Response: Correct the invalid value.

ICE266A

ICE266A ddname TRLUPD COLUMN OVERLAPS RECORD DESCRIPTOR WORD

Explanation: Critical. For variable-length record processing, the TRLUPD parameter of the OUTFIL data set associated with ddname specified an item that overlapped the record descriptor word (RDW). Only data bytes, which start at position 5 for variable-length records, can be overlaid.

The error is one of the following:

- c: was not specified for the first item so the default of 1: was used for that item. Example:

  ```
  TRLUPD=(COUNT=(M10,LENGTH=5))
  ```

- c: was specified for an item with a value for c which was less than 5. Example:

  ```
  TRLUPD=(3:COUNT=(M10,LENGTH=5))
  ```

System Action: The program terminates.

Programmer Response: Specify c: with a value of 5 or more for the first item. Ensure that c is 5 or more for any other c: values you specify. Example:

```
  TRLUPD=(8:COUNT=(M10,LENGTH=5),
    25:TOT=(5,4,ZD,EDIT=(IIIIT)))
```

ICE659I

ICE659I RECORDS RESIZED FROM n BYTES TO m BYTES

Explanation: For this RESIZE operator, n indicates the original size of the input records, and m indicates the resulting size of the output records.

System Action: None.

Programmer Response: None.

ICE660A

ICE660A FROM DATA SET MUST HAVE FIXED LENGTH RECORDS

Explanation: Critical. For this RESIZE operator, the input data set associated with the FROM ddname has variable-length records rather than fixed-length records.

System Action: This operation is terminated.

Programmer Response: Specify a FROM ddname associated with a fixed-length data set (for example, FB or F).

ICE661A

ICE661A TOLEN MUST NOT BE THE SAME AS FROM DATA SET RECORD LENGTH

Explanation: Critical. For this RESIZE operator, the output length specified by TOLEN is the same as:

- the reformatted record length if INREC is used.
- the record length of the input data set associated with the FROM ddname if INREC is not used.

A RESIZE operation requires a FROM record length different from TOLEN.

System Action: This operation is terminated.

Programmer Response: Specify a TOLEN value that is larger or smaller than the FROM record length.

Changed Messages

This section shows existing messages that have been changed significantly for PTFs UK90025 and UK90026. Refer to *z/OS DFSORT Messages, Codes and Diagnosis Guide (SC26-7525-05)* for general information on DFSORT messages.

ICE018A, ICE113A, ICE114A

These messages will be issued for the TRLID operand of OUTFIL for the same reasons they are issued for the INCLUDE operand of OUTFIL.

ICE107A

This message will be issued for the KEYBEGIN operand of IFTHEN for the same reasons it is issued for the BEGIN operand of IFTHEN.

ICE111A

This message will be issued for the following additional situations:

- The length (m) for a KEYBEGIN field was greater than 256 bytes.
- The length (m) for a HEX or TRAN=HEX field was greater than 16376 bytes.
- The length (m) for a TRAN=BIT field was greater than 4094 bytes.

ICE151A, ICE221A

These messages will be issued for the TRLID operand of OUTFIL for the same reasons they are issued for the BEGIN operand of OUTFIL.

ICE189A

This message will be issued for the following additional situations if Blockset could not be used:

- JPn"string" in EXEC PARM (where n is 0-9)
- p,m,TRAN=keyword
- p,TRAN=keyword
- DATE5
- ICETOOL called DFSORT for RESIZE processing.

ICE211I

ICE211I OLD OUTFIL STATEMENT PROCESSING USED

Explanation: This OUTFIL statement did not have any of the following operands: FNAMES, FILES, STARTREC, ENDREC, SAMPLE, INCLUDE, OMIT, SAVE, PARSE, OUTREC, BUILD, VTOF, CONVERT, VLFILL, OVERLAY, IFTHEN, FTOV, VLTRIM, REPEAT, SPLIT, SPLITBY, SPLIT1R, NULLOFL, LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, NODETAIL, BLKCCH1, BLKCCH2, BLKCCT1, REMOVECC, ACCEPT or IFTRAIL. For compatibility, this OUTFIL statement was treated as an "old" OUTFIL statement and all of its operands were ignored.

System Action: None. Processing continues, but OUTFIL data sets are not associated with this OUTFIL statement. If the Blockset technique is not selected, control statement errors could result from continuation of this OUTFIL statement.

Programmer Response: None, unless this is not an old OUTFIL statement, in which case valid operands from the list above should be specified.

ICE214A

This message will be issued for the following additional situations:

- For the KEYBEGIN operand of IFTHEN for the same reasons it is issued for the BEGIN operand of IFTHEN.
- If IFTRAIL is specified with HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, NODETAIL, LINES, SPLIT, SPLITBY, SPLIT1R, REPEAT, FTOV, VTOF, VLTRIM or VLFILL.

ICE223A

This message will be issued for the following changed situation:

- The length for an input field was greater than 32752 bytes.

ICE276A

The following are added to the list of reserved words for Symbols: DATE5, ADDDAYS, SUBDAYS, ADDMONS, SUBMONS, ADDYEARS, SUBYEARS, DATEDIFF, LASTDAYW, LASTDAYM, LASTDAYQ, LASTDAYY, NEXTDday and PREVDday (where day is SUN, MON, TUE, WED, THU, FRI or SAT).

ICE288I

ICE288I INPUT OR OUTPUT DATE VALUE OUT OF RANGE FOR DATE CONVERSION OR DATE ARITHMETIC

Explanation: For a date conversion operation using TOJUL, TOGREG, WEEKDAY, DT or DTNS, or for a date arithmetic operation using ADDDAYS, SUBDAYS, ADDMONS, SUBMONS, ADDYEARS, SUBYEARS, DATEDIFF, LASTDAYW, LASTDAYM, LASTDAYQ, LASTDAYY, NEXTDday or PREVDday (where day is SUN, MON, TUE, WED, THU, FRI or SAT), an invalid input date was used, or an invalid output date would have been produced. A date value is considered invalid if any of the following range conditions are not met:

- yy must be between 00 and 99
- ccyy must be between 0001 and 9999
- mm must be between 01 and 12
- dd must be between 01 and 31, and must be valid for the year and month
- ddd must be between 001 and 366 for a leap year, or between 001 and 365 for a non-leap year.

A date is also considered invalid if the input field is a CH/ZD special indicator of binary zeros, blanks or binary ones, and the output field is PD, or if an input field for DATEDIFF is a special indicator.

System Action: Asterisks are printed for each invalid output value. The message is only issued once. Processing continues.

Programmer Response: Check for output values containing asterisks and ensure that the input date value is valid and that you are not converting a CH/ZD special indicator of binary zeros, blanks or binary ones to a PD value.

ICE613A

This message will be issued for the following additional situations:

- For the RESIZE operator if FROM, TO or TOLEN is not specified.
- For the COUNT operator if RC4, RC8 or RC12 is specified, and EMPTY, NOTEMPTY, HIGHER, LOWER, EQUAL or NOTEQUAL is not specified.

ICE614A

RESIZE is now a valid operator.

ICE623A

The maximum number of HEADER and ON operands for DISPLAY is now 50 (was 20).

ICE624A

This message will be issued for the following additional situation:

- For the RESIZE operator if more than one TO operand is specified.

ICE637A

ICE637A ddname RECORD LENGTH OF n BYTES EXCEEDS MAXIMUM WIDTH OF m BYTES

Explanation: Critical.

- For a DISPLAY or OCCUR operator without NOCC: The calculated record length for the indicated list data set was greater than 8192 with LONGLINE, greater than 2048 without LONGLINE, or greater than the maximum width specified by WIDTH(m). n is the total bytes required in the list data set record for the carriage control character, the title lines (resulting from specified title elements), column widths (resulting from specified ON, HEADER, PLUS, BLANK, TOTAL, BREAK, BTITLE, and BTOTAL operands), and blanks before and between title elements and columns (resulting from specified INDENT, TBETWEEN, BETWEEN, and STATLEFT operands). m is the value specified for the WIDTH operand, or the maximum allowed value if WIDTH was not specified.
- For a DISPLAY or OCCUR operator with NOCC:
  - WIDTH(8192) was specified. This exceeds the maximum width of 8191 allowed with NOCC. n is 8192 and m is 8191.
  - The calculated record length for the indicated list data set was greater than 8191 with LONGLINE, greater than 2047 without LONGLINE, or greater than the maximum width specified by WIDTH(m). n is the total bytes required in the list data set record for the title lines (resulting from specified title elements), column widths (resulting from specified ON, HEADER, PLUS, BLANK, TOTAL, BREAK, BTITLE, and BTOTAL operands), and blanks before and between title elements and columns (resulting from specified INDENT, TBETWEEN, BETWEEN, and STATLEFT operands). m is the value specified for the WIDTH operand, or the maximum allowed value if WIDTH was not specified.
- For a COUNT operator with WRITE: The calculated record length for the indicated output data set was greater than the maximum width specified by WIDTH(m). n is the total bytes required in the output data set record for the count line (resulting from specified TEXT, DIGITS and EDCOUNT operands). m is the value specified for the WIDTH operand.

System Action: The operation is terminated.

Programmer Response:

- For a DISPLAY or OCCUR operator:

  If WIDTH is specified, remove the WIDTH operand and let ICETOOL set the width automatically, or if you need to set the WIDTH explicitly, increase its value to n or greater.

  If WIDTH is not specified, use the LONGLINE operand to increase the maximum width to 8192 without NOCC or 8191 with NOCC.

  If m is 8192 without NOCC or 8191 with NOCC, take one or more of the following actions:

  - Use formatting items or the PLUS or BLANK operand. For example, use ON(21,18,ZD,U19) instead of ON(21,18,ZD) with TOTAL to change the column width from 32 bytes to 20 bytes.
  - Reduce the length of one or more HEADER strings.
  - Reduce the length of one or more ON fields. For example, if an ON(1,8,PD) field always has zeros in bytes 1 through 3, use instead (1,8,PD,U09), or ON(4,5,PD) with BLANK, to reduce the column width from 16 bytes to 10 bytes.
  - Reduce the number of ON fields, especially if the BTOTAL or TOTAL operand is used.
  - Reduce BETWEEN(n).
  - Reduce INDENT(n).

  – Remove STATLEFT.

  – Reduce TBETWEEN(n).

- For a COUNT operator, remove the WIDTH operand and let ICETOOL set the width, or if you need to set the WIDTH explicitly, increase its value to n or greater.

ICE646A

ICE646A RECORD COUNT MEETS CRITERIA - RC=nn SET

Explanation: Critical. EMPTY, NOTEMPTY, HIGHER(n), LOWER(n), EQUAL(n), or NOTEQUAL(n) was specified for this COUNT operator. Because the record count met the specified criteria, RC=08 (if RC8 was specified) or RC=12 (if RC12 was specified or defaulted) was set for this COUNT operation.

System Action: This operation is terminated.

Programmer Response: None.

ICE652A

RESIZE is added to the list of ICETOOL operators for which this message is issued. A DFSORT OUTREC statement must not be used with RESIZE.