User Guide for DFSORT PTF UK90013

July, 2008

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 V1R5 PTF **UK90013**, which was first made available in **July, 2008**.

This PTF provides important enhancements to DFSORT and DFSORT's 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 multi-part 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.

Contents

User Guide for DFSORT PTF UK90013

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 V1R5 PTF **UK90013**, which was first made available in **July, 2008**, provides important enhancements to DFSORT and DFSORT's 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 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 V1R5 PTF UK90013. 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* 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* for general information on DFSORT messages.

Summary of Changes

Find and replace

FINDREP is a new option that allows you to do various types of find and replace operations on your records. FINDREP makes it easy to replace character or hexadecimal input constants anywhere in your records with character, hexadecimal or null output constants. For input and output constants of different lengths, bytes after the

replaced constants will be shifted left or right, as appropriate. For fixed-length records, blanks will be filled in on the right as needed. For variable-length records, the record length will be changed as needed.

FINDREP can be used in an INREC, OUTREC or OUTFIL statement, or in an IFTHEN clause, in the same way BUILD and OVERLAY can be used.

Various options of FINDREP allow you to define one or more input constants and a corresponding output constant (IN, OUT), define one or more pairs of input and output constants (INOUT), start and end the find scan at specified positions (STARTPOS, ENDPOS), stop after a specified number of constants are replaced (DO), increase or decrease the length of the output record (MAXLEN), define the action to be taken if nonblank characters overrun the end of the record (OVERRUN), and specify whether output constants are to replace or overlay input constants (SHIFT).

DFSORT symbols can be used for constants specified with FINDREP.

Group operations

WHEN=GROUP is a new type of IFTHEN clause that allows you to do various types of operations involving groups of records. WHEN=GROUP makes it easy to propagate fields from the first record of a group to the other records of the group, add an identifier to each record of the group, or add a 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 groups, including or omitting groups, and so on.

WHEN=GROUP can be used in IFTHEN clauses in an INREC, OUTREC or OUTFIL statement in the same way WHEN=INIT can be used.

Various options of WHEN=GROUP allow you to use logical expressions to define the beginning and end of a group (BEGIN, END), define the number of records in a group (RECORDS), and define the fields, identifiers and sequence numbers to be added to the records of each group (PUSH).

DFSORT symbols can be used for columns, fields and constants specified with WHEN=GROUP clauses.

DATASORT

DATASORT is a new operator of ICETOOL that allows you to sort data records between header (first) records and trailer (last) records. DATASORT makes it easy to sort the data records while keeping one or more header records and/or one or more trailer records in place. DATASORT does not require an "identifier" in the header or trailer records; it can treat the first n records as header records and the last n records as trailer records.

Various options of DATASORT allow you to define the number of header records and/or trailer records (HEADER or FIRST, TRAILER or LAST), the ddname for the input data set (FROM), the ddname for the output data set (TO), and the SORT and other DFSORT control statements to be used for the DATASORT operation (USING).

DFSORT symbols can be used for the number of header and trailer records specified with DATASORT.

SUBSET

SUBSET is a new operator of ICETOOL that allows you to create a subset of the input or output records by specifying that you want to keep or remove header (first) records, trailer (last) records, or records with specific relative record numbers. SUBSET makes it easy to keep or remove records based on these criteria. SUBSET does not require an "identifier" or "sequence number" in the records to be kept or removed.

Various options of SUBSET allow you to define the criteria (HEADER or FIRST, TRAILER or LAST, RRN), the ddname for the input data set (FROM), the ddname for the output data set to contain the records that meet the

criteria and/or don't meet the criteria (TO, DISCARD), whether the records that meet the criteria are to be kept or removed (KEEP, REMOVE), whether the criteria are to be applied to the input or output records (INPUT, OUTPUT), and DFSORT control statements to be used for the SUBSET operation (USING).

DFSORT symbols can be used for the number of header and trailer records and for the relative record numbers specified with SUBSET.

SELECT first n duplicates

ICETOOL's SELECT operator now allows you to select the first n records with each key or the first n duplicate records with each key. New FIRST(n) and FIRSTDUP(n) options make it easy to select records representing "top" and "bottom" categories (for example, the top 5 students in each class).

DFSORT symbols can be used for n with FIRST(n) or FIRSTDUP(n).

SPLICE with non-blank fields

ICETOOL's SPLICE operator now allows you to create a single record for each key by splicing the base record with every specified nonblank field from each overlay record. A new WITHANY option makes it easy to collect information from multiple records with the same key. You can now do a splice involving duplicate records with nonconsecutive or missing WITH fields, something that could not be accomplished previously with the existing WITHEACH option.

DISPLAY with count

ICETOOL's DISPLAY operator now allows you to display counts in reports. New COUNT('string'), EDCOUNT(formatting), BCOUNT('string') and EDBCOUNT(formatting) options make it easy to print overall record count and break record count statistics in various forms in a report, similar to the existing statistics for a report (overall total, maximum, minimum and average and break total, maximum, minimum and average).

DFSORT symbols can be used for 'string' with COUNT('string') and BCOUNT('string').

DISPLAY/OCCUR with multiple and multipart titles

ICETOOL's DISPLAY and OCCUR operators now allow you to display up to three title lines, each composed of up to three strings. The enhanced TITLE('string1','string2','string3') option makes it easy to use multiple strings for each title, including a combination of inline constants, and constants from DFSORT symbols including system information. The use of up to three TITLE options makes it easy to display multiline titles.

A new TLEFT option allows you to left justify the title lines instead of centering them. A new TFIRST option allows you to only display the title lines on the first page of the report instead of on every page of the report.

DFSORT symbols can be used for 'string1', 'string2' and 'string3' with TITLE('string1','string2','string3').

DISPLAY/OCCUR without carriage control

ICETOOL's DISPLAY and OCCUR operators now allow you to create reports without carriage control characters and with RECFM=FB instead of RECFM=FBA. A new NOCC option makes it easy to suppress the carriage control character. With NOCC, a blank line is used instead of a page eject control character to separate elements of the report.

COUNT in output record

ICETOOL's COUNT operator now allows you to create a count data set with an output record containing the record count. New WRITE(countdd), TEXT('string'), DIGITS(n) and EDCOUNT(formatting) options make it easy to create an output data set with a record containing text and the record count in various forms.

DFSORT symbols can be used for 'string' with TEXT('string').

COUNT with add/subtract

ICETOOL's COUNT operator now allows you to add a value to, or subtract a value from, the record count. New ADD(n) and SUB(n) options make it easy to increase or decrease, respectively, the actual record count to get a resulting modified record count. This is especially useful for dealing with data sets that contain header and/or trailer records.

The resulting modified record count is displayed in the count message in TOOLMSG and in the count data set, and used to determine if the criteria specified by the existing EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) option is satisfied.

DFSORT symbols can be used for n with ADD(n) and SUB(n).

BLKSIZE default for input DUMMY

DFSORT will no longer terminate for a SORTIN DD DUMMY or SORTINnn DD DUMMY statement with RECFM and LRECL, but no BLKSIZE. Instead, DFSORT will use an an appropriate BLKSIZE to process the DUMMY data set successfully.

Note: If DFSORT's Blockset technique is not selected, DFSORT may still terminate for a SORTIN DD DUMMY or SORTINnn DD DUMMY statement with RECFM and LRECL, but no BLKSIZE.

SKIP=0L default for SECTIONS

DFSORT will no longer terminate when an OUTFIL SECTIONS field is not followed by a keyword (SKIP, HEADER3, TRAILER3). Instead, DFSORT will use a default keyword of SKIP=0L to process the sections successfully with no blank lines between sections on the same page.

DFSORT symbols can be used for section fields.

SORTOUT=ddname default for FNAMES

DFSORT will now use the ddname specified by a SORTOUT=ddname option in DFSPARM, the ddname specified by a SORTOUT=ddname option in a parameter list, or the ddname specified in a TO(ddname) option of an ICETOOL operator, as the default ddname for an OUTFIL statement without a FNAMES or FILES option.

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:

- Prior to this PTF, an ICETOOL job with an operator (for example, SELECT) that uses TO(ddname) and USING(xxxx) with //ddname and //xxxxOUT DD statements and an OUTFIL statement without FNAMES or FILES, would treat the ddname data set as a SORTOUT data set and the xxxxOUT data set as the OUTFIL data set.

With this PTF, the same job will treat the ddname data set as the OUTFIL data set and ignore the xxxxOUT data set. If you want to treat the xxxxOUT data set as the OUTFIL data set, change your ICETOOL operator to use TO(xxxxOUT).

- Prior to this PTF, an OUTFIL statement with FTOV and IFOUTLEN=n would set the LRECL of the OUTFIL data set and the length of each OUTFIL record to n.

  With this PTF, the same situation will result in setting the LRECL of the OUTFIL data set and the length of each record to n+4. If you want to set the LRECL of the OUTFIL data set and the length of each OUTFIL record to n, change IFOUTLEN to specify n-4.

Find and replace

Introduction

FINDREP and IFTHEN FINDREP are new INREC, OUTREC and OUTFIL operands that allow you to do various types of find and replace operations on your records. FINDREP gives you new capabilities for finding character or hexadecimal input constants anywhere in your records and replacing them with character, hexadecimal or null output constants. As appropriate, bytes can be shifted left or right, blank padding can be added for fixed-length records, and the length can be changed for variable-length records.

Various options of FINDREP allow you to define one or more input constants and a corresponding output constant, define one or more pairs of input and output constants, start and end the find scan at specified positions, stop after a specified number of constants are replaced, increase or decrease the length of the output record, define the action to be taken if nonblank characters overrun the end of the record, and specify whether output constants are to replace or overlay input constants.

As a simple example, you could use the following INREC statement to replace all instances of 'Goodbye' in your input records with 'Bye', shift the bytes after the replaced constants to the left, and pad on the right with blanks.

```
INREC FINDREP=(IN=C'Goodbye',OUT=C'Bye')
```

If you had 60 byte input records like this:

```
*"Goodbye John"*"Goodbye William"*"Goodbye Goodboy"*
"Goodbye Michael""Good Dog""Goodbye Goodbye"
```

you would get 60-byte output records like this:

```
*"Bye John"*"Bye William"*"Bye Goodboy"*
"Bye Michael""Good Dog""Bye Bye"
```

Syntax

The syntax for the FINDREP operand is as follows:

```
FINDREP=(input/output_constants,STARTPOS=p,ENDPOS=q,DO=n,MAXLEN=n,
   OVERRUN=ERROR/TRUNC,SHIFT=YES/NO)
```

Detailed Description

You can use FINDREP to find constants anywhere in a record and replace them with other constants of the same or different lengths.

You can use FINDREP and IFTHEN FINDREP in the INREC, OUTREC and OUTFIL statements.

input/output constants

Input and output constants for find and replace processing are defined as follows:

- **input constant:** An input constant can be specified as a single character string, a repeated character string, a single hexadecimal string, or a repeated hexadecimal string. The syntax is: C'string', nC'string', X'string' or nX'string'. n can be 1 to 256. The string can be 1 to 256 characters, or 1 to 256 pairs of hexadecimal digits. The total length of the constant must not exceed 256 bytes. Use two apostrophes for a single apostrophe.

  DFSORT symbols can be used for C'string' and X'string' (but not for nC'string' or nX'string').

  output constant: An output constant can be specified as a null string, a single character string, a repeated character string, a single hexadecimal string, or a repeated hexadecimal string. The syntax is: C'' (null), C'string', nC'string', X'string' or nX'string'. n can be 1 to 256. The string can be a null, or 1 to 256 characters, or 1 to 256 pairs of hexadecimal digits. The total length of the constant must not exceed 256 bytes. Use two apostrophes for a single apostrophe. A null string can be used to remove input constants.

  DFSORT symbols can be used for C'string' and X'string' (but not for nC'string', nX'string' or C'').

You must specify the input and output constants to be used for find and replace processing in one of the following ways. The default processing for each method described below can be changed with various options described later.

- **IN=incon,OUT=outcon**

  Specifies one input constant and one output constant. Position 1 (for fixed-length records) or 5 (for variable-length records) will be set as the current position. The current position of the input record will be checked for the input constant. If a match is not found at the current position, the current position will be incremented by 1, and the process will be repeated. If a match is found at the current position, the output constant will replace the input constant, the current position will be incremented past the input constant, and the process will be repeated. Bytes after the replaced constants up to the end of the record will be shifted left or right as needed. Processing will stop when the current position is beyond the end of the input record.

  Example:

  ```
  INREC IFTHEN=(WHEN=(11,1,CH,EQ,C'3'),
          FINDREP=(IN=C'YES',OUT=C'NO'))
  ```

  Replaces every C'YES' input constant in records with a C'3' in position 11 with a C'NO' output constant, and shifts the bytes after each replaced constant to the left.

- **IN=(incon1,incon2,...,inconx),OUT=outcon**

  Specifies multiple input constants and one output constant. Position 1 (for fixed-length records) or 5 (for variable-length records) will be set as the current position. The current position of the input record will be checked for each input constant in turn until a match is found or all of the input constants have been checked. If a match is not found at the current position for any input constant, the current position will be incremented by 1, and the process will be repeated. If a match is found at the current position, the output constant will replace the input constant, the current position will be incremented past the input constant, and the process will be repeated. Bytes after the replaced constants up to the end of the record will be shifted left or right as needed. Processing will stop when the current position is beyond the end of the input record.

  Example:

  ```
  OUTREC FINDREP=(IN=(X'FF',3X'00'),OUT=C'')
  ```

  Removes every X'FF' and X'000000' input constant, and shifts the bytes after each removed constant to the left.

- **INOUT=(incon1,outcon1,incon2,outcon2,...,inconx,outconx)**

  Specifies one or more pairs of input and output constants. Position 1 (for fixed-length records) or 5 (for variable-length records) will be set as the current position. The current position of the input record will be

checked for each input constant in turn until a match is found or all of the input constants have been checked. If a match is not found at the current position for any input constant, the current position will be incremented by 1, and the process will be repeated. If a match is found at the current position for an input constant, the corresponding output constant will replace the input constant, the current position will be incremented past the input constant, and the process will be repeated. Bytes after the replaced constants up to the end of the record will be shifted left or right as needed. Processing will stop when the current position is beyond the end of the input record.

Example:

```
OUTFIL FINDREP=(INOUT=(C'SAT',C'SATURDAY',C'SUN',C'SUNDAY'))
```

Replaces every C'SAT' input constant with a C'SATURDAY' output constant, and every C'SUN' input constant with a C'SUNDAY' output constant, and shifts the bytes after each replaced constant to the right.

Optional operands:

You can specify any or all of the following options to change the default processing described earlier:

- **STARTPOS=p**

  Specifies the starting position in the input record for the find scan, overriding the default of position 1 for a fixed-length record or position 5 for a variable-length record. Use STARTPOS=p if you want to start your find scan at a particular position. p can be 1 to 32752. If p is less than 5 for a variable-length record, 5 will be used for p. If p is beyond the end of the input record, find and replace processing will not be performed for the record.

  Example:

  ```
  INREC FINDREP=(IN=C'Yes',OUT=C'YES',STARTPOS=11)
  ```

  Replaces every C'Yes' input constant found starting at or after position 11 with a C'YES' output constant.

- **ENDPOS=q**

  Specifies the ending position in the input record for the find scan, overriding the default of the end of the record. Use ENDPOS=q if you want to end your find scan at a particular position. ENDPOS=q only applies to the end of the find scan; bytes will still be shifted up to the end of the record as needed. q can be 1 to 32752. If q is less than 5 for a variable-length record, 5 will be used for q. If q is beyond the end of the input record, the end of the record will be used for q. If STARTPOS=p and ENDPOS=q are both specified, and p is greater than q, find and replace processing will not be performed for the record.

  Example:

  ```
  OUTREC FINDREP=(IN=(C'D27',C'A52',C'X31'),OUT=C'INVALID',
     ENDPOS=2015)
  ```

  Replaces every C'D27', C'A52' and C'X31' input constant found before or at position 2015 with a C'INVALID' output constant, and shifts the bytes after each replaced constant to the right (past 2015 up to the end of the record).

- **DO=n**

  Specifies the maximum number of times find and replace is to be performed for a record, overriding the default of every time. Scanning for the input constant stops when n input constants have been found and replaced. Use DO=n if you want to stop after a particular number of constants have been replaced. n can be 1 to 1000.

  Example:

  ```
  OUTFIL FNAMES=OUT1,FINDREP=(IN=X'015C',OUT=X'015D',DO=3)
  ```

  Replaces the first three X'015C' input constants found with X'015D' output constants.

- **MAXLEN=n**

  Specifies the maximum length to be used for the output record created by find and replace processing, overriding the default of using the maximum length of the input record. Use MAXLEN=n if you want to increase or decrease the output record length. (For IFTHEN FINDREP, MAXLEN=n can only be used to increase the output record length, not decrease it. See "Notes" below for more information.)

  If an output constant is larger than a corresponding input constant, MAXLEN=n can be used to increase the size of the output record to allow for shifting characters to the right. n can be 1 to 32752. MAXLEN=n will be used to set the LRECL of the output data set, when appropriate.

  Example:

  ```
  INREC FINDREP=(INOUT=(C'01',C'January',C'02',C'February',
    C'03',C'March'),MAXLEN=150)
  ```

  Replaces every C'01' input constant with a C'January' output constant, every C'02' input constant with a C'February' output constant, and every C'03' input constant with a C'March' output constant, and shifts the bytes after each replaced constant to the right, allowing the record to expand to 150 bytes.

- **OVERRUN=ERROR or OVERRUN=TRUNC**

  Specifies the action DFSORT is to take if an overrun occurs, that is:

  – if nonblank bytes are shifted to the right past the end of the output record as specified by MAXLEN=n, or as defaulted to the input record length, or

  – if MAXLEN=n is used to make the output record smaller than the input record, and nonblank bytes are found in the input record past the end of the output record.

  OVERRUN=ERROR is the default; it tells DFSORT to issue an error message and terminate if an overrun occurs.

  Use OVERRUN=TRUNC if you want DFSORT to truncate the output record to the MAXLEN=n or input record length if an overrun occurs, rather than terminating. Bytes beyond the end of the output record are lost.

  Example:

  ```
  OUTREC FINDREP=(IN=X'FFFF',OUT=C'INVALID',MAXLEN=50,OVERRUN=TRUNC)
  ```

  Replaces every X'FFFF' input constant with a C'INVALID' output constant, and shifts the bytes after the replaced constants to the right. 50 byte output records are created. Bytes shifted past position 50 are lost. Without OVERRUN=TRUNC, if nonblank characters were shifted past position 50, an overrun error message would be issued and the job would terminate.

- **SHIFT=YES or SHIFT=NO**

  Specifies the action DFSORT is to take if an input constant is to be replaced by an output constant of a different length.

  SHIFT=YES is the default; it tells DFSORT to shift the bytes after each replaced input constant to the left or right as needed.

  Use SHIFT=NO if you want DFSORT to overlay the input constant with the output constant instead of shifting bytes. If a matching input constant is found and the output constant is smaller than the input constant, the current position will be incremented past the output constant rather than past the input constant before processing continues.

  Example:

  ```
  OUTREC FINDREP=(IN=(C'KW1=YES,',C'KW1=NO,'),OUT=C'KW2',SHIFT=NO)
  ```

  Replaces every C'KW1=YES,' input constant with a C'KW2=YES,' output constant. Replaces every C'KW1=NO,' input constant with a C'KW2=NO,' output constant. SHIFT=NO ensures that C'KW1' is replaced

with C'KW2' and the '=YES,' and '=NO,' bytes are kept. Without SHIFT=NO, the '=YES,' and '=NO,' bytes would be removed.

Notes:

- FINDREP cannot be specified with BUILD or OVERLAY in an IFTHEN clause.
- FINDREP cannot be specified with BUILD, OVERLAY, IFTHEN or IFOUTLEN in an INREC or OUTREC statement.
- FINDREP cannot be specified with BUILD, OVERLAY, IFTHEN, IFOUTLEN, VTOF, CONVERT or VLFILL in an OUTFIL statement.
- For a single FINDREP operand, when a constant is replaced at the current position, no further checks are performed at that position. So a single FINDREP cannot be used to change a constant and then change it again. For example, if you had an input record with:

  ```
  ABC
  ```

  and used this INREC statement:

  ```
  INREC FINDREP=(INOUT=(C'AB',C'XY',C'XYC',C'RST'))
  ```

  the output record would be:

  ```
  XYC
  ```

  After C'AB' is changed to C'XY', the current pointer is advanced to C'C' so C'XYZ' is not found. However, since each IFTHEN clause starts at the beginning of the record, multiple IFTHEN clauses with FINDREP can be used to change a constant and then change it again. If you used:

  ```
  INREC IFTHEN=(WHEN=INIT,FINDREP=(INOUT=(C'AB',C'XY'))),
        IFTHEN=(WHEN=INIT,FINDREP=(INOUT=(C'XYC',C'RST')))
  ```

  for the ABC record, the output would be:

  ```
  RST
  ```

  The first IFTHEN clause would change C'AB' to C'XY'. The second IFTHEN clause would start over from the first position and change C'XYC' to C'RST'.
- Duplicates and supersets of the same input constant after the first are effectively ignored, whereas subsets of the same input constant after the first are processed. For example, if you had this input record:

  ```
  ABCD ABQ
  ```

  and you used this INREC statement:

  ```
  INREC FINDREP=(INOUT=(C'AB',C'XY',C'ABCD',C'RSTU'))
  ```

  the output record would be:

  ```
  XYCD XYQ
  ```

  Since C'AB' is changed to C'XY', C'ABCD' is not found. If you wanted to change C'ABCD' to C'RSTU' and other instances of C'AB' to C'XY', you would need to specify C'ABCD' first. If you used this INREC statement:

  ```
  INREC FINDREP=(INOUT=(C'ABCD',C'RSTU',C'AB',C'XY'))
  ```

  the output record would be:

  ```
  RSTU XYQ
  ```

  If C'ABCD' is found, it is changed to C'RSTU'. Otherwise, if C'AB' is found, it is changed to C'XY'.

- For fixed-length records, FINDREP in an IFTHEN clause operates against the maximum record padded with blanks on the right as needed. If a blank constant is being replaced, the blanks on the right will be replaced. For example, if we have an 80 byte FB input record and use:

```
INREC IFTHEN=(WHEN=INIT,BUILD=(1,20)),
      IFTHEN=(WHEN=(2,1,CH,EQ,C'R'),
        FINDREP=(IN=C' ',OUT=C'ABC'))
```

  Although in this case BUILD=(1,20) will result in a 20-byte output record, IFTHEN FINDREP will operate against the maximum 80-byte record padded with 60 blanks on the right. For a record with 'R' in position 2, each of those 60 blanks on the right will be replaced with C'ABC'. This will cause an overrun of the 80 byte record, thus resulting in termination (OVERRUN=ERROR) or truncation (OVERRUN=TRUNC). ENDPOS=20 could be used to stop FINDREP from replacing the 60 blanks on the right.

  Note that if we have an 80-byte input record with 'ABC' in positions 1-3 and we use:

```
INREC IFTHEN=(WHEN=INIT,BUILD=(1,3)),
      IFTHEN=(WHEN=INIT,FINDREP=(IN=C'ABC',OUT=C'12345'))
```

  an overrun will not occur because IFTHEN FINDREP will operate against an 80-byte record padded with 77 blanks on the right, rather than against a 3-byte record even though the output record will be 3 bytes.

- For FINDREP in an IFTHEN clause, MAXLEN=n can be used to increase the maximum output length, but cannot be used to decrease the maximum output length. For example, with:

```
OUTREC IFTHEN=(WHEN=INIT,BUILD=(1,100)),
       IFTHEN=(WHEN=INIT,
         FINDREP=(IN=C'A',OUT=C'B',MAXLEN=150))
```

  the output length will be set to 150, whereas with:

```
OUTREC IFTHEN=(WHEN=INIT,BUILD=(1,100)),
       IFTHEN=(WHEN=INIT,
         FINDREP=(IN=C'A',OUT=C'B',MAXLEN=70))
```

  the output length will be set to 100.

Example 1

```
SORT FIELDS=(1,2,CH,A)
OUTREC FINDREP=(IN=C'*',OUT=C' ')
```

This example illustrates how you can replace a character with another character anywhere in FB or VB records.

The FB input records might look like this:

```
05 ***** JUNE ***************
02         * APRIL *
01* * * * DAISY * * * *
03BETTY ************
```

We want to replace every asterisk with a blank. We use IN=C'*' to indicate we want to find each asterisk, and OUT=C' ' to indicate we want to replace it with a blank.

The sorted output records look like this:

```
01        DAISY
02           APRIL
03BETTY
05       JUNE
```

Example 2

```
  OPTION COPY
  INREC FINDREP=(IN=(X'00',X'FF'),OUT=C'')
```

This example illustrates how you can remove characters from FB or VB records.

The VB input records might look like this in hexadecimal:

```
RDW----|Data
000F0000D1E4D5C5**00**C1D7D9C9D3**FF**
00100000C2C5E3E3E8**00**C4C1C9E2E8**FF**
```

We want to remove each X'00' and X'FF' character. We use IN=(X'00',X'FF') to indicate we want to find each X'00' and X'FF' character, and OUT=C'' (null) to indicate we want to remove it.

The output records look like this:

```
RDW----|Data
000D0000D1E4D5C5C1D7D9C9D3
000E0000C2C5E3E3E8C4C1C9E2E8
```

Note that the X'00' and X'FF' characters have been removed and the RDW length decreased accordingly. For VB input records, FINDREP processing automatically starts at position 5 after the RDW so the X'00' characters in the RDW are **not** affected.

Example 3

```
 OPTION COPY
 OUTFIL FINDREP=(INOUT=(C'AM',C'IN THE MORNING',
   C'PM',C'IN THE EVENING'),MAXLEN=70)
```

This example illustrates how you can replace values in FB or VB records with larger values and shift the rest of the bytes to the right.

The 40 byte FB input records might look like this:

```
COFFEE AT 12:28 AM, TOAST AT 06:15 PM
MILK AT 03:17 PM, BAGELS AT 05:03 PM
PUDDING AT 09:32 AM
```

We want to replace each instance of 'AM' with 'IN THE MORNING' and each instance of 'PM' with 'IN THE EVENING' and shift the bytes after 'AM' or 'PM' to the right. We use INOUT to indicate find and replace pairs. Since replacing the smaller values with the larger values can cause the remaining bytes to be shifted beyond the end of the 40 byte record, we use MAXLEN to set the output record to 70 bytes to allow for the expansion, overriding the default of using the input length for the output record.

The 70 byte FB output records look like this:

```
COFFEE AT 12:28 IN THE MORNING, TOAST AT 06:15 IN THE EVENING
MILK AT 03:17 IN THE EVENING, BAGELS AT 05:03 IN THE EVENING
PUDDING AT 09:32 IN THE MORNING
```

Example 4

```
  OPTION COPY
  INREC FINDREP=(IN=C'BALANCE',
                OUT=C'BALANCE 1000',SHIFT=NO,DO=1)
```

This example illustrates how you can find a value in FB or VB records and overlay it with a larger value without shifting other bytes in the records.

The FB input records might look like this:

```
CUSTOMER1 10100
MNTHLY STMT BALANCE 2000
CUSTOMER2 11100
MNTHLY STMT REQUIRES NO MODIFICATION ACCT BALANCE 5000
CUSTOMER3 11111
YOUR INFO ENCLOSED
MNTHLY STMT REQUIRES MODIFICATION ACCT BALANCE 7000
```

We want to replace every instance of 'BALANCE dddd' with 'BALANCE 1000' where dddd can be any value. We use IN=C'BALANCE' to indicate we want find each instance of 'BALANCE'. We use OUT='BALANCE 1000' to indicate we want to replace it with 'BALANCE 1000'. We use SHIFT=NO to do an overlay, overriding the default of shifting bytes. Since we only have at most one instance of 'BALANCE dddd' in a record, we can use DO=1 to stop processing a record after one replacement of 'BALANCE dddd' with 'BALANCE 1000'. In this case, DO=1 is more efficient than the default of continuing to look for more instances of 'BALANCE dddd' after the first instance. We would get the same result without DO=1, just less efficiently.

The output records look like this:

```
CUSTOMER1 10100
MNTHLY STMT BALANCE 1000
CUSTOMER2 11100
MNTHLY STMT REQUIRES NO MODIFICATION ACCT BALANCE 1000
CUSTOMER3 11111
YOUR INFO ENCLOSED
MNTHLY STMT REQUIRES MODIFICATION ACCT BALANCE 1000
```

Group operations

Introduction

WHEN=GROUP is a new type of IFTHEN clause that allows you to do various types of operations involving groups of records. WHEN=GROUP gives you new capabilities for propagating fields, identifiers and sequence numbers within groups, and further facilitates other types of group operations such as sorting by groups, including or omitting records by groups, and so on. WHEN=GROUP clauses can be used in INREC, OUTREC and OUTFIL statements by themselves or in conjunction with the other existing types of IFTHEN clauses.

Various options of the WHEN=GROUP clause allow you to define the beginning and/or end of a group using a simple or complex logical expression, define the number of records in a group, and define fields, identifiers and sequence numbers to be added to the records in each group.

As a simple example, you could use the following statement to define groups of 3 records and add an identifier and sequence number to each record of each group.

```
INREC IFTHEN=(WHEN=GROUP,RECORDS=3,PUSH=(15:ID=3,19:SEQ=5))
```

Each group consists of 3 records. Positions 15-17 of each record are overlaid by a 3-byte identifier that increments by 1 for each group. Positions 19-23 of each record are overlaid by a 5-byte sequence number that increments by 1 for each record and restarts at 1 for each group. If the input records were as follows:

```
Vicky
Sri Hari
Frank
David
Dave
Regina
Sam
Viet
```

The output records would be:

```
Vicky         001 00001
Sri Hari      001 00002
Frank         001 00003
David         002 00001
Dave          002 00002
Regina        002 00003
Sam           003 00001
Viet          003 00002
```

As another example, you could use the following statements to define groups of 20 byte records and sort the groups by a date in the first record of each group.

```
  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)
```

Each group begins with a record containing C'DATE:' in positions 1-5. The date in positions 7-14 of that record is propagated to positions 21-28 of each record in its group. We then sort the records in each group by the propagated date. Finally, we remove the propagated date. If the input records were as follows:

```
DATE: 20080612
DRILL   520
SAW     250
WRENCH  005
DATE: 20080601
HAMMER  008
SAW     123
DATE: 20080527
WRENCH  302
```

The output records would be:

```
DATE: 20080527
WRENCH  302
DATE: 20080601
HAMMER  008
SAW     123
DATE: 20080612
DRILL   520
SAW     250
WRENCH  005
```

Syntax

The syntax for the IFTHEN WHEN=GROUP clause is as follows:

```
IFTHEN=(WHEN=GROUP,BEGIN=(logexp),END=(logexp),RECORDS=n,
  PUSH=(c:item,...))
```

Detailed Description

You can use IFTHEN WHEN=GROUP clauses to identify groups of records in various ways and propagate fields, identifiers and sequence numbers to the records of each group. Fields, identifiers and sequence numbers are not propagated to records before, between or after the identified groups.

You can use IFTHEN WHEN=GROUP clauses in the INREC, OUTREC and OUTFIL statements along with the other IFTHEN clauses. WHEN=GROUP and WHEN=INIT clauses can be intermixed, but must be specified before WHEN=(logexp), WHEN=ANY or WHEN=NONE clauses. All IFTHEN clauses will be processed in the order specified. WHEN=GROUP clauses, like WHEN=INIT clauses, are processed for every record.

WHEN=GROUP must be specified to indicate this is a WHEN=GROUP clause.

You can specify the BEGIN, END and RECORDS operands in any combination to define the groups, but you must specify at least one of these operands.

- **BEGIN=(logexp)**

  Specifies the criteria to be tested to determine if a record starts a group. See the discussion of the INCLUDE statement in *z/OS DFSORT Application Programming Guide* for details of the logical expressons you can use. You can specify all of the logical expressions for BEGIN 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. Instead, missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

  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.

  A new group starts with a record that satisfies the BEGIN condition, that is, when the specified logical expression is true for that record. If BEGIN is specified without END or RECORDS, all of the records from the begin record up to but not including the next begin record belong to a group. Here's an example of groups with BEGIN=(1,1,CH,EQ,C'A'):

  ```
  H
  R
  A group 1
  B group 1
  C group 1
  A group 2
  A group 3
  B group 3
  ```

  Example:

  ```
  OUTREC IFTHEN=(WHEN=GROUP,
    BEGIN=(1,40,SS,EQ,C'J82',OR,1,40,SS,EQ,C'M72'),
    PUSH=(41:ID=5))
  ```

Starts a new group each time C'J82' or 'M72' is found anywhere in positions 1-40 of a record. Overlays positions 41-45 of each record of a group with a 5-byte ZD identifier. The records before the first group are not changed.

- **END=(logexp)**

  Specifies the criteria to be tested to determine if a record ends a group. See the discussion of BEGIN=(logexp) above for details of the logical expressions you can use for END=(logexp).

  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.

  A group ends whenever a record satisfies the END condition, that is, whenever the specified logical expression is true for that record. If END is specified without BEGIN, all of the records up to the end record (or the last record of the data set) belong to a group. Here's an example of groups with

  ```
  END=(1,1,CH,EQ,C'T')

  A group 1
  B group 1
  T group 1
  T group 2
  A group 3
  T group 3
  M group 4
  ```

  If END is specified with BEGIN, all of the records from the begin record up to the end record (or the last record of the data set) belong to a group. Here's an example of groups with

  ```
  BEGIN=(1,1,CH,EQ,C'H'),END=(1,1,CH,EQ,C'T')

  H group 1
  B group 1
  T group 1
  T
  H group 2
  T group 2
  M
  N
  H group 3
  A group 3
  ```

  Example:

  ```
  OUTREC IFTHEN=(WHEN=GROUP,
            BEGIN=(1,2,CH,EQ,C'02',AND,8,3,CH,EQ,C'YES'),
            END=(11,5,CH,EQ,C'PAGE:'),PUSH=(61:SEQ=3))
  ```

  Starts a new group each time C'02' is found in positions 1-2 and C'YES' is found in positions 8-10 of a record. Ends the group when C'PAGE:' is found in positions 11-15 of a record. Overlays positions 61-63 of each record of a group with a 3-byte ZD sequence number. The records between the groups are not changed.

- **RECORDS=n**

  Specifies the maximum number of records in a group. n can be 1 to 2000000000.

  If RECORDS is specified without BEGIN or END, a new group starts after n records. Here's an example of groups with

```
RECORDS=3

H group 1
B group 1
T group 1
H group 2
B group 2
T group 2
M group 3
N group 3
```

If RECORDS is specified with BEGIN, up to n records starting with the begin record belong to a group. Here's an example of groups with

```
BEGIN=(1,1,CH,EQ,C'H'),RECORDS=3

H group 1
B group 1
T group 1
A
H group 2
B group 2
H group 3
M group 3
N group 3
P
```

The records between and after the groups are not changed.

If RECORDS is specified with END, the group ends after up to n records or with the end record (or the last record of the data set), whichever comes first. Here's an example of groups with

```
BEGIN=(1,1,CH,EQ,C'H'),END=(1,1,CH,EQ,C'T'),RECORDS=4

H group 1
B group 1
T group 1
A
H group 2
B group 2
C group 2
D group 2
E
H group 3
M group 3
```

The records between the groups are not changed.

You must specify the PUSH operand to define at least one field, identifier or sequence number to be added to the records of each group. You can specify any combination or number of fields, identifiers and sequence numbers.

PUSH=(c:item,...)

Specifies the position where each field, identifier or sequence number is to be overlaid in the records of each group.

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

You can use the following in PUSH:

- **c:**

  Specifies the output position (column) to be overlaid. 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 can specify items in any order and overlap output columns. c can be 1 to 32752.

  If you specify an item that extends the output record beyond the end of the input record, the record length is automatically increased to that length, and blanks are filled in on the left as needed. For variable-length records, the RDW length is increased to correspond to the larger record length after all of the items are processed. Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

  DFSORT symbols can be used for c:.

- **p,m**

  Specifies a field in the first input record of each group to be propagated to every record of the group. p specifies the starting position of the field in the input record and m specifies its length. A field must not extend beyond position 32752.

  DFSORT symbols can be used for p,m.

- **ID=n**

  Specifies a ZD identifier of length n is to be added to every record of each group. The identifier starts at 1 for the first group and is incremented by 1 for each subsequent group. n can be 1 to 15.

- **SEQ=n**

  Specifies a ZD sequence number of length n is to be added to every record of each group. The sequence number starts at 1 for the first record of each group and is incremented by 1 for each subsequent record of the group. n can be 1 to 15.

Example 1

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

This example illustrates how you can INCLUDE groups of FB records between a header and a trailer. We add an ID after the end of each record to indicate whether it's part of a group or not, INCLUDE on the ID, and then remove it.

The 30-byte FB input records might look like this:

```
C33  Not in a group
HDR   Start Group 1
A01    Group 1 record
B02    Group 1 record
C03    Group 1 record
TRL   End Group 1
R24  Not in a group
T02  Not in a group
HDR   Start Group 2
D04    Group 2 record
E05    Group 2 record
TRL   End Group 2
F97  Not in a group
```

In the output data set we only want to include groups of records that start with 'HDR' and end with 'TRL'.

We use an IFTHEN WHEN=GROUP clause to put a non-blank character in each record that is part of a group. BEGIN indicates a group starts with a record that has 'HDR' in positions 1-3. END indicates a group ends with a record that has 'TRL' in positions 1-3. PUSH overlays a 1-byte ID character at position 31 in each record of a group (after the end of the record). After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
C33  Not in a group
HDR   Start Group 1           1
A01    Group 1 record         1
B02    Group 1 record         1
C03    Group 1 record         1
TRL   End Group 1             1
R24  Not in a group
T02  Not in a group
HDR   Start Group 2           2
D04    Group 2 record         2
E05    Group 2 record         2
TRL   End Group 2             2
F97  Not in a group
```

Note that the records within a group have a non-blank character in position 31 whereas the records outside groups have a blank character in position 31. The ID starts at 1 for the first group and is incremented by 1 for each subsequent group. Since we are only allowing one character for the ID, when the ID counter gets to 10, a '0' will appear in position 31. That's fine since we are just looking for a non-blank to indicate a record within a group, or a blank to indicate a record outside of a group.

We use an OUTFIL statement to only INCLUDE records with a non-blank in position 31, and to remove the ID character so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
HDR   Start Group 1
A01    Group 1 record
B02    Group 1 record
C03    Group 1 record
TRL   End Group 1
HDR   Start Group 2
D04    Group 2 record
E05    Group 2 record
TRL   End Group 2
```

Example 2

```
OPTION COPY
INREC IFTHEN=(WHEN=INIT,BUILD=(1,4,6:5)),
  IFTHEN=(WHEN=GROUP,BEGIN=(6,3,CH,EQ,C'HDR'),
    END=(6,3,CH,EQ,C'TRL'),PUSH=(5:ID=1))
OUTFIL INCLUDE=(5,1,CH,NE,C' '),BUILD=(1,4,5:6)
```

This example illustrates how you can INCLUDE groups of VB records between a header and a trailer. It's similar to Example 1, but here the records are variable-length. For the FB records, we could add the ID after the end of each record and then remove it without changing the records. But we can't add the ID at the end of each VB record because that would pad all of the records to a fixed length. So, instead we insert the ID between the RDW and the first data byte of each record, and later remove it.

The VB input records might look like this:

```
Len|Data
 23|C33  Not in a group
 23|HDR   Start Group 1
 25|A01    Group 1 record
 25|B02    Group 1 record
 25|C03    Group 1 record
 21|TRL   End Group 1
 23|R24  Not in a group
 23|T02  Not in a group
 23|HDR   Start Group 2
 25|D04    Group 2 record
 25|E05    Group 2 record
 21|TRL   End Group 2
 25|F97  Not in a group
```

In the output data set we only want to include groups of records that start with 'HDR' and end with 'TRL'.

We use an IFTHEN WHEN=INIT clause to reformat each record so it has room for the ID byte between the RDW and the first data byte. After the WHEN=INIT clause is executed, the intermediate records look like this:

```
Len|Data
 24| C33  Not in a group
 24| HDR   Start Group 1
 26| A01    Group 1 record
 26| B02    Group 1 record
 26| C03    Group 1 record
 22| TRL   End Group 1
 24| R24  Not in a group
 24| T02  Not in a group
 24| HDR   Start Group 2
 26| D04    Group 2 record
 26| E05    Group 2 record
 22| TRL   End Group 2
 26| F97  Not in a group
```

Note that position 5 is blank and the 'HDR' and 'TRL' characters have been shifted over to positions 6-8.

We use an IFTHEN WHEN=GROUP clause to put a non-blank character in each record that is part of a group. BEGIN indicates a group starts with a record that has 'HDR' in positions 6-8. END indicates a group ends with a record that has 'TRL' in positions 6-8. PUSH overlays a 1-byte ID character at position 5 in each record of a group. After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
Len|Data
 24| C33  Not in a group
 24|1HDR   Start Group 1
 26|1A01    Group 1 record
 26|1B02    Group 1 record
 26|1C03    Group 1 record
 22|1TRL   End Group 1
 24| R24  Not in a group
 24| T02  Not in a group
 24|2HDR   Start Group 2
 26|2D04    Group 2 record
 26|2E05    Group 2 record
 22|2TRL   End Group 2
 26| F97  Not in a group
```

Note that the records within a group have a non-blank character in position 5 whereas the records outside groups have a blank character in position 5. The ID starts at 1 for the first group and is incremented by 1 for each

subsequent group. Since we are only allowing one character for the ID, when the ID counter gets to 10, a '0' will appear in position 5. That's fine since we are just looking for a non-blank to indicate a record within a group, or a blank to indicate a record outside of a group.

We use an OUTFIL statement to only INCLUDE records with a non-blank in position 5, and to remove the ID character so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
 23|HDR   Start Group 1
 25|A01    Group 1 record
 25|B02    Group 1 record
 25|C03    Group 1 record
 21|TRL   End Group 1
 23|HDR   Start Group 2
 25|D04    Group 2 record
 25|E05    Group 2 record
 21|TRL   End Group 2
```

Example 3

```
  INREC IFTHEN=(WHEN=GROUP,BEGIN=(2,4,CH,EQ,C'RPT.'),
    PUSH=(31:6,8))
  OPTION EQUALS
  SORT FIELDS=(31,8,CH,A)
  OUTFIL INCLUDE=(31,8,CH,EQ,C'FRANK',OR,
     31,8,CH,EQ,C'SRIHARI'),BUILD=(1,30)
```

This example illustrates how you can SORT and INCLUDE groups of FB records depending on a value in the first record of each group. We propagate the value in the first record of the group to every record of the group, SORT and INCLUDE on the value, and then remove it.

The 30-byte FBA input records might look like this:

```
1RPT.SRIHARI
 LINE  1 FOR REPORT 1
 LINE  2 FOR REPORT 1
 ...
1RPT.VICKY
 LINE  1 FOR REPORT 2
 LINE  2 FOR REPORT 2
 ...
1RPT.FRANK
 LINE  1 FOR REPORT 3
 LINE  2 FOR REPORT 3
 ...
1RPT.DAVID
 LINE  1 FOR REPORT 4
 LINE  2 FOR REPORT 4
 ...
```

Each report starts with 'RPT.reptname' in positions 2-13. In the output data set we only want to include records for reports with specific reptname values, and the reptname values we want can change from run to run. We also want to sort by the reptname values in ascending order. For this example, let's say we just want the SRIHARI and FRANK reports.

We use an IFTHEN WHEN=GROUP clause to propagate the reptname value to each record of the group. BEGIN indicates a group starts with 'RPT.' in positions 2-5. PUSH overlays the reptname value from the first record of the

group (the 'RPT.reptname' record) at positions 31-38 (after the end of the record) in each record of the group including the first. After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
1RPT.SRIHARI                  SRIHARI
 LINE  1 FOR REPORT 1         SRIHARI
 LINE  2 FOR REPORT 1         SRIHARI
 ...                          SRIHARI
1RPT.VICKY                    VICKY
 LINE  1 FOR REPORT 2         VICKY
 LINE  2 FOR REPORT 2         VICKY
 ...                          VICKY
1RPT.FRANK                    FRANK
 LINE  1 FOR REPORT 3         FRANK
 LINE  2 FOR REPORT 3         FRANK
 ...                          FRANK
1RPT.DAVID                    DAVID
 LINE  1 FOR REPORT 4         DAVID
 LINE  2 FOR REPORT 4         DAVID
 ...                          DAVID
```

Note that the records of each group have the reptname value from the first record of that group in positions 31-38.

We use a SORT statement to sort ascending on the reptname in positions 31-38. We use the EQUALS option to ensure that records in the same group (that is, with the same reptname value) are kept in their original order. After the SORT statement is executed, the intermediate records look like this:

```
1RPT.DAVID                    DAVID
 LINE  1 FOR REPORT 4         DAVID
 LINE  2 FOR REPORT 4         DAVID
 ...                          DAVID
1RPT.FRANK                    FRANK
 LINE  1 FOR REPORT 3         FRANK
 LINE  2 FOR REPORT 3         FRANK
 ...                          FRANK
1RPT.SRIHARI                  SRIHARI
 LINE  1 FOR REPORT 1         SRIHARI
 LINE  2 FOR REPORT 1         SRIHARI
 ...                          SRIHARI
1RPT.VICKY                    VICKY
 LINE  1 FOR REPORT 2         VICKY
 LINE  2 FOR REPORT 2         VICKY
 ...                          VICKY
```

We use an OUTFIL statement to only INCLUDE the records with a a reptname of FRANK or SRIHARI in positions 31-38, and to remove the reptname from positions 31-38 so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
1RPT.FRANK
 LINE  1 FOR REPORT 3
 LINE  2 FOR REPORT 3
 ...
1RPT.SRIHARI
 LINE  1 FOR REPORT 1
 LINE  2 FOR REPORT 1
 ...
```

Example 4

```
  OPTION COPY
  OUTREC IFTHEN=(WHEN=INIT,BUILD=(1,4,8:5)),
    IFTHEN=(WHEN=GROUP,BEGIN=(8,5,CH,EQ,C'PAGE:'),
    PUSH=(5:SEQ=3))
  OUTFIL INCLUDE=(5,3,ZD,EQ,2,OR,5,3,ZD,EQ,3),
    BUILD=(1,4,5:8)
```

This example illustrates how you can INCLUDE specific relative records from groups of VB records. We insert a sequence number between the RDW and the first data byte of each record. The sequence number restarts at 1 for the first record in each group. We INCLUDE on the sequence number and then remove it.

The VB input records might look like this:

```
Len|Data
 12|PAGE:  1
 22|LINE 1 OF REPORT A
 22|LINE 2 OF REPORT A
 22|LINE 3 OF REPORT A
 22|LINE 4 OF REPORT A
   |...
 12|PAGE:  2
 23|LINE 66 OF REPORT A
 23|LINE 67 OF REPORT A
 23|LINE 68 OF REPORT A
 23|LINE 69 OF REPORT A
   |...
 12|PAGE:  3
 24|LINE 131 OF REPORT A
 24|LINE 132 OF REPORT A
 24|LINE 133 OF REPORT A
 24|LINE 134 OF REPORT A
   |...
```

We use an IFTHEN WHEN=INIT clause to reformat each record so it has room for the 3-byte sequence number between the RDW and the first data byte. After the WHEN=INIT clause is executed, the intermediate records look like this:

```
Len|Data
 15|   PAGE:  1
 25|   LINE 1 OF REPORT A
 25|   LINE 2 OF REPORT A
 25|   LINE 3 OF REPORT A
 25|   LINE 4 OF REPORT A
   |   ...
 15|   PAGE:  2
 26|   LINE 66 OF REPORT A
 26|   LINE 67 OF REPORT A
 26|   LINE 68 OF REPORT A
 26|   LINE 69 OF REPORT A
   |   ...
 15|   PAGE:  3
 27|   LINE 131 OF REPORT A
 27|   LINE 132 OF REPORT A
 27|   LINE 133 OF REPORT A
 27|   LINE 134 OF REPORT A
```

Note that positions 5-7 are blank and the 'PAGE:' characters have been shifted over to positions 8-12.

We use an IFTHEN WHEN=GROUP clause to put a 3-byte sequence number in each record. BEGIN indicates a group starts with a record that has 'PAGE:' in positions 8-12. PUSH overlays a 3-byte sequence number at positions 5-7 of each record. The sequence number starts at 1 for the first record of a group and is incremented by 1 for each subsequent record of the group. After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
Len|Data
 15|001PAGE:  1
 25|002LINE 1 OF REPORT A
 25|003LINE 2 OF REPORT A
 25|004LINE 3 OF REPORT A
 25|005LINE 4 OF REPORT A
   |006...
 15|001PAGE:  2
 26|002LINE 66 OF REPORT A
 26|003LINE 67 OF REPORT A
 26|004LINE 68 OF REPORT A
 26|005LINE 69 OF REPORT A
   |006...
 15|001PAGE:  3
 27|002LINE 131 OF REPORT A
 27|003LINE 132 OF REPORT A
 27|004LINE 133 OF REPORT A
 27|005LINE 134 OF REPORT A
   |006...
```

Note that the records in each group are numbered starting with 001 for the first record of each group. The records we want have sequence numbers 002 and 003.

We use an OUTFIL statement to only INCLUDE the records with sequence number 002 or 003, and to remove the sequence numbers so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
Len|Data
 22|LINE 1 OF REPORT A
 22|LINE 2 OF REPORT A
 23|LINE 66 OF REPORT A
 23|LINE 67 OF REPORT A
 24|LINE 131 OF REPORT A
 24|LINE 132 OF REPORT A
```

DATASORT

Introduction

DATASORT is a new ICETOOL operator that allows you to sort the data records in a data set without sorting the header or trailer records. DATASORT gives you new capabilities for sorting the data records between header records (first n records of the data set) and trailer records (last n records of the data set) while keeping the header and trailer records in place. DATASORT does not require an "identifier" in the header or trailer records; it can treat the first n records as header records and the last n records as trailer records.

Various options of DATASORT allow you to define the ddname for the input data set, the ddname for the output data set, the number of header records and/or trailer records, and the SORT and other DFSORT control statements to be used for the DATASORT operation.

As an example, you could use the following DATASORT operator to sort the data records by positions 1-16 while keeping the header record as the first record and the two trailer records are the last two records, respectively.

```
...
//TOOLIN DD *
DATASORT FROM(IN) TO(OUT) HEADER TRAILER(2) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,16,CH,A)
```

If the IN data set contained these records:

```
2008/04/23
Geometry
Algebra
Trigonometry
Calculus
0004
End of data set
```

the OUT data set would contain these records:

```
2008/04/23
Algebra
Calculus
Geometry
Trigonometry
0004
End of data set
```

Syntax

The syntax for the DATASORT operator is as follows:

```
DATASORT FROM(indd) TO(outdd) HEADER|FIRST|HEADER(x)|FIRST(x)
 TRAILER|LAST|TRAILER(x)|LAST(x) USING(xxxx) VSAMTYPE(x)
```

Detailed Description

DATASORT **copies** one or more header records and one or more trailer records to the output data set in their original input record order, while **sorting** the data records between the header and trailer records. By definition, header records are the first n records in the input data set, the trailer records are the last n records in the input data set, and the data records (also called detail records) are the records between the header and trailer records.

DFSORT is called to copy the header and trailer records and to sort the data records. DFSORT uses its E15 and E35 exits to process the records as needed. You must supply a DFSORT SORT statement in the xxxxCNTL data set to indicate the control fields to be used for sorting the data records. ICETOOL passes the EQUALS option to DFSORT to ensure that duplicates are kept in their original input order.

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 DATASORT operator in the same way as for the existing operators. No operation specific values are returned for DATASORT.

You must specify the FROM(indd), TO(outdd) and USING(xxxx) operands.

You must specify a header operand (HEADER, FIRST, HEADER(x), FIRST(x)) or a trailer operand (TRAILER, LAST, TRAILER(x), LAST(x)). If you specify a header operand without a trailer operand, only the header records will be kept in place. If you specify a trailer operand without a header operand, only the trailer records will be kept

in place. If you specify a header operand and a trailer operand, both the header records and trailer records will be kept in place.

You can only specify one header operand. You can only specifiy one trailer operand.

The VSAMTYPE(x) operand is optional.

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.

- **HEADER or FIRST**

  Specifies one header record (the first record in the indd data set) is to be kept in place.

  HEADER and FIRST are equivalent to HEADER(1) and FIRST(1).

- **HEADER(x) or FIRST(x)**

  Specifies x header records (the first x records in the indd data set) are to be kept in place. x must be specified as n or +n where n can be 1 to 1000000.

  DFSORT symbols can be used for n and +n in HEADER(x) and FIRST(x).

- **TRAILER or LAST**

  Specifies one trailer record (the last record in the indd data set) is to be kept in place.

  TRAILER and LAST are equivalent to TRAILER(1) and LAST(1).

- **TRAILER(x) or LAST(x)**

  Specifies x trailer records (the last x records in the indd data set) are to be kept in place. x must be specified as n or +n where n can be 1 to 1000000.

  DFSORT symbols can be used for n and +n in TRAILER(x) and LAST(x).

- **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. 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.

  You must observe these rules for the control statements in the xxxxCNTL data set:

  – A SORT statement must be present

  – MODS and OUTREC statements should not be present.

  – A STOPAFT operand should not be present.

  – Comment statements can be present.

- Header and trailer records will only be affected by the SKIPREC option and OUTFIL statements. SKIPREC=n will remove the first n indd records, so the first header record will be the n+1 indd record. OUTFIL statements will process the header and trailer records in the normal way.
- Data records will be processed by INCLUDE, OMIT, INREC, SUM, OPTION and OUTFIL statements in the normal way.
- If you use INREC to change the length of the data records, DFSORT will preserve the header and trailer records by setting the TO data set LRECL to the maximum of the input or reformatted record length. For fixed-length records, DFSORT will pad the header and trailer records, or data records, on the right with blanks as appropriate.
- You can further process the outdd records **after** DATASORT processing using an OUTFIL statement like this:

  ```
  OUTFIL FNAMES=outdd,...
  ```

  or multiple OUTFIL statements like this:

  ```
  OUTFIL FNAMES=outdd,...
  OUTFIL FNAMES=outdd1,...
  ```

  For example, with TO(OUT1) you could further modify the OUT1 records after DATASORT processing, with a statement like this:

  ```
  OUTFIL FNAMES=OUT1,REMOVECC,
    TRAILER1=('Record count ',COUNT=(M11,LENGTH=5))
  ```

- **VSAMTYPE(x)**

  Specifies the record type for a VSAM input data set. x must be either F for fixed-length record processing or V for variable-length record processing.

  If VSAMTYPE(x) is specified, ICETOOL will pass a RECORD TYPE=x control statement to DFSORT. (If you specify a RECORD TYPE=x statement in the xxxxCNTL data set, it will override the one passed by ICETOOL.)

Example 1

```
DATASORT FROM(INPUT) TO(OUTPUT) -
  HEADER TRAILER USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(16,13,CH,A)
```

This example illustrates how you can sort the data records between a header record (first record) and a trailer record (last record) in an FB or VB data set.

The FB input records might look like this:

```
MM9999900510100823DDDDD FFFFF 004200806128
AAR   FIRST C  1134341444441     XXXXXXXXX
ATX   SECOND   7777777770111     XXXXXXXXX
ATX   THIRD L  6297132201111     XXXXXXXXX
ATX   FOURTH   2830012906356     XXXXXXXXX
MM9999900510100823DDDDD FFFFF 004
```

We want to keep the MM records in place and sort the other records by the CH field in positions 16-28. We use HEADER and TRAILER to indicate the first and last records should not be sorted. We use the SORT statement to SORT ascending on positions 16-28.

The output records look like this:

```
MM9999900510100823DDDDD FFFFF 004200806128
AAR   FIRST C  1134341444441     XXXXXXXXX
ATX   FOURTH   2830012906356     XXXXXXXXX
ATX   THIRD L  6297132201111     XXXXXXXXX
ATX   SECOND   7777777770111     XXXXXXXXX
MM9999900510100823DDDDD FFFFF 004
```

Example 2

```
DATASORT FROM(IN) TO(OUT) -
  HEADER(2) TRAILER(3) USING(CTL1)
//CTL1CNTL DD *
  INREC IFTHEN=(WHEN=(24,2,CH,EQ,C'23'),
    OVERLAY=(30:C'Old'))
  SORT FIELDS=(1,14,CH,A)
  OUTFIL FNAMES=OUT,
    IFTHEN=(WHEN=(24,2,CH,EQ,C'23'),
      OVERLAY=(35:C'First'))
```

This example illustrates how you can sort the data records between header records (first records) and trailer records (last records) in an FB or VB data set, and modify just the data records or the header, data and trailer records.

The FB input records might look like this:

```
Header 1       2008/04/23
Header 2       2008/04/23
Geometry       2008/04/24
Algebra        2008/04/23
Trigonometry   2008/04/24
Calculus       2008/04/25
Geography      2008/04/25
History        2008/04/23
Trailer 1      2008/04/23
Trailer 2      2008/04/23
Trailer 3      2008/04/23
```

We want to keep the two Header records and the three Trailer records in place and sort the other records by the CH field in positions 1-14. We want to put 'Old' in positions 30-32 of each **data record** (but not the Header or Trailer records) that has '23' in positions 24-25. We want to put 'First' in positions 35-39 of **each record** (Header, data and Trailer) that has '23' in positions 24-25.

We use HEADER(2) and TRAILER(3) to indicate the first two records and last 3 records should not be sorted. We use the INREC statement to add 'Old' to data records that have '23' in positions 24-25. INREC applies to the data records, but not to the Header and Trailer records. We use the SORT statement to sort ascending on positions 1-14. We use the OUTFIL statement to add 'First' to Header, data and Trailer records that have '23' in positions 24-25. OUTFIL applies to all of the records.

The output records look like this:

```
Header 1       2008/04/23         First
Header 2       2008/04/23         First
Algebra        2008/04/23    Old  First
Calculus       2008/04/25
Geography      2008/04/25
Geometry       2008/04/24
History        2008/04/23    Old  First
Trigonometry   2008/04/24
Trailer 1      2008/04/23         First
Trailer 2      2008/04/23         First
Trailer 3      2008/04/23         First
```

SUBSET

Introduction

SUBSET is a new ICETOOL operator that allows you to create a subset of the input or output records with specific header, trailer, and relative records, or without specific header, trailer, and relative records. SUBSET gives you new capabilities for keeping or removing the first n records of your data set, the last n records of your data set, and/or specific relative records in your data set. SUBSET does not require an "identifier" in the records to be kept or removed; it keeps track of the first n records, relative record numbers, and the last n records automatically.

Various options of SUBSET allow you to define the criteria for keeping or removing records, the ddname for the input data set, the ddname for the output data set to contain the records that meet the criteria, the ddname for the output data set to contain the records that don't meet the criteria, whether the records that meet the criteria are to be kept or removed, whether the criteria are to be applied to the input or output records, and DFSORT control statements to be used for the SUBSET operation.

As an example, you could use the following SUBSET operator to keep the first two input records, the fifth, sixth and seventh input records, and the last input record.

```
SUBSET FROM(IN) TO(OUT) KEEP INPUT FIRST(2) RRN(5,7) LAST
```

If the IN data set contained these records:

```
MASTER03.IN
2008/04/23
Vicky
Frank
Regina
Viet
David
Dave
Carrie
Sam
Sri Hari
Martin
UPDATE03.OUT
```

the OUT data set would contain these records:

```
MASTER03.IN
2008/04/23
Regina
Viet
David
UPDATE03.OUT
```

Syntax

The syntax for the SUBSET operator is as follows:

```
SUBSET FROM(indd) TO(outdd) DISCARD(savedd) KEEP|REMOVE INPUT|OUTPUT
 HEADER|FIRST|HEADER(x)|FIRST(x)
 RRN(x)|RRN(x,y)|RRN(x,*) ...
 TRAILER|LAST|TRAILER(x)|LAST(x)
 USING(xxxx) VSAMTYPE(x)
```

Detailed Description

SUBSET keeps or removes input or output records based on meeting criteria for the first n records, specific relative record numbers, and the last n records. DFSORT writes the records that are kept or not removed to the outdd data set. DISCARD(savedd) can be used to write the records that are removed or not kept to the savedd data set.

DFSORT is called to copy or sort the indd data set, as appropriate. ICETOOL uses its E15 or E35 exit to determine which records to include in the outdd or savedd data set. ICETOOL passes the EQUALS option to DFSORT to ensure that duplicates are kept in their original input order if records are sorted.

If the criteria includes the last n records, ICETOOL may call DFSORT twice. For the first pass, ICETOOL counts the indd records without opening the output data sets. For the second pass, ICETOOL opens the output data sets and does SUBSET processing against the indd data set using the count obtained in the first pass.

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 SUBSET operator in the same way as for the existing operators. No operation specific values are returned for SUBSET.

You must specify the FROM(indd), TO(outdd) or DISCARD(savedd), KEEP or REMOVE, and INPUT or OUTPUT operands.

If you do not specify a header operand (HEADER, FIRST, HEADER(x), FIRST(x)), a relative record number operand (RRN(x), RRN(x,y), RRN(x,*)), or a trailer operand (TRAILER, LAST, TRAILER(x), LAST(x)), all of the records will be kept or removed. You can specify a header operand, relative record number operands, and a trailer operand in any combination. Records will be kept or removed according to the criteria you specify.

You can only specify one header operand. You can specify from 1 to 300 relative record number operands in any combination. You can only specify one trailer operand.

The USING(xxxx) and VSAMTYPE(x) operands are optional.

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 which DFSORT will write the records it selects for the operation (that is, the records that are kept or not removed according to the specified criteria). If TO(outdd) is specified, an outdd DD statement must be present.

  TO and DISCARD can both be specified. If DISCARD is not specified, TO must be specified. If TO is not specified, DISCARD must be specified.

  The ddname specified in the TO operand must not be the same as the ddname specified in the FROM or DISCARD operand.

- **DISCARD(savedd)**

  Specifies the ddname of the output data set to which DFSORT will write the records it does not select for the operation. If DISCARD(savedd) is specified, a savedd DD statement must be present.

  TO and DISCARD can both be specified. If DISCARD is not specified, TO must be specified. If TO is not specified, DISCARD must be specified.

  The ddname specified in the DISCARD operand must not be the same as the ddname specified in the FROM or TO operand.

- **KEEP**

  Specifies that the records that meet the criteria are to be kept.

- **REMOVE**

  Specifies that the records that meet the criteria are to be removed.

- **INPUT**

  Specifies that the criteria are to be applied using the first n input records, relative input record numbers, and the last n input records. Use INPUT if you want to apply the criteria directly to the records from the indd data set.

  The criteria will be applied to keep or remove records before they are reformatted by INREC, sorted by SORT and summed by SUM. The kept records will subsequently be reformatted, sorted and summed. OUTFIL will be applied to the resulting records.

  As an example, if the input data set contains these records:

```
AAAA R01
AAAA R02
BBBB R03
CCCC R04
CCCC R05
CCCC R06
DDDD R07
DDDD R08
EEEE R09
EEEE R10
EEEE R11
```

  and the following SUBSET operator and DFSORT control statements were specified:

```
...
//TOOLIN DD *
SUBSET FROM(IN) TO(OUT) KEEP INPUT RRN(3,10) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,5,CH,D)
  SUM FIELDS=NONE
/*
```

First, input records 3-10 would be kept, so the intermediate result would be:

```
BBBB R03
CCCC R04
CCCC R05
CCCC R06
DDDD R07
DDDD R08
EEEE R09
EEEE R10
```

Then the SORT statement would be applied, so the intermediate result would be:

```
EEEE R09
EEEE R10
DDDD R07
DDDD R08
CCCC R04
CCCC R05
CCCC R06
BBBB R03
```

Finally the SUM statement would be applied, so the final output in the OUT data set would be:

```
EEEE R09
DDDD R07
CCCC R04
BBBB R03
```

- **OUTPUT**

  Specifies that the criteria are to be applied using the first n output records, relative output record numbers, and the last n output records. Use OUTPUT if you want to apply the criteria to the indd records after they are processed by INREC, SORT and SUM as specified.

  The criteria will be applied to keep or remove records after they are reformatted by INREC, sorted by SORT and summed by SUM. OUTFIL will be applied to the resulting records.

  As an example, if the input data set contains these records:

```
AAAA R01
AAAA R02
BBBB R03
CCCC R04
CCCC R05
CCCC R06
DDDD R07
DDDD R08
EEEE R09
EEEE R10
EEEE R11
```

  and the following SUBSET operator and DFSORT control statements were specified:

```
...
//TOOLIN DD *
SUBSET FROM(IN) TO(OUT) KEEP OUTPUT RRN(2,3) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,5,CH,D)
  SUM FIELDS=NONE
/*
```

  First, the SORT statement would be applied, so the intermediate result would be:

```
EEEE R09
EEEE R10
EEEE R11
DDDD R07
DDDD R08
CCCC R04
CCCC R05
CCCC R06
BBBB R03
AAAA R01
AAAA R02
```

Then the SUM statement would be applied, so the intermediate result would be:

```
EEEE R09
DDDD R07
CCCC R04
BBBB R03
AAAA R01
```

Finally, output records 2-3 would be kept, so the final output in the OUT data set would be:

```
DDDD R07
CCCC R04
```

- **HEADER or FIRST**

  Specifies one header record (the first record) is to be kept or removed.

  HEADER and FIRST are equivalent to HEADER(1) and FIRST(1).

- **HEADER(x) or FIRST(x)**

  Specifies x header records (the first x records) are to be kept or removed. For example, HEADER(3) or FIRST(3) keeps or removes the first three records. x must be specified as n or +n where n can be 1 to 999999999999999.

  DFSORT symbols can be used for n and +n in HEADER(x) and FIRST(x).

- **RRN(x)**

  Specifies relative record number x is to be kept or removed. For example, RRN(8) keeps or removes the eighth record. x must be specified as n or +n where n can be 1 to 999999999999999.

  DFSORT symbols can be used for n and +n in RRN(x).

- **RRN(x,y)**

  Specifies relative record numbers x through y are to be kept or removed. x can be less than, equal to, or greater than y. For example, RRN(5,10) and RRN(10,5) both keep or remove the fifth through tenth records. x and y must be specified as n or +n where n can be 1 to 99999999999999.

  DFSORT symbols can be used for n and +n in RRN(x,y).

- **RRN(x,*)**

  Specifies relative record numbers x through the last record are to be kept or removed. For example, RRN(7,*) keeps or removes the seventh through last records. x must be specified as n or +n where n can be 1 to 99999999999999.

  DFSORT symbols can be used for n and +n in RRN(x,*).

- **TRAILER or LAST**

  Specifies one trailer record (the last record) is to be kept or removed.

TRAILER and LAST are equivalent to TRAILER(1) and LAST(1).

- **TRAILER(x) or LAST(x)**

  Specifies x trailer records (the last x records) are to be kept or removed. For example, TRAILER(4) or LAST(4) keeps or removes the last 4 records. x must be specified as n or +n where n can be 1 to 999999999999999.

  DFSORT symbols can be used for n and +n in TRAILER(x) and LAST(x).

- **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 specify a SORT statement in xxxxCNTL and you want to override dynamic allocation of work data sets for this operation, you can use xxxxWKdd DD statements for that purpose.

  You must observe these rules for the control statements in the xxxxCNTL data set:

  - MODS and OUTREC statements should not be present.
  - SKIPREC and STOPAFT operands, and INCLUDE and OMIT statements, should not be present.
  - A SORT statement can be present unless INPUT and DISCARD(savedd) are specified.
  - INREC and SUM statements can be present.
  - If INPUT is specified, the records selected will not be affected by INREC, SORT or SUM. If OUTPUT is specified, the records selected will be affected by INREC, SORT and SUM.
  - Comment statements can be present.
  - If you specify TO(outdd) without DISCARD(savedd), you can further process the outdd records **after** SUBSET processing using an OUTFIL statement like this:

    ```
    OUTFIL FNAMES=outdd,...
    ```

    or multiple OUTFIL statements like this:

    ```
    OUTFIL FNAMES=outdd,...
    OUTFIL FNAMES=outdd1,...
    ```

  - If you specify DISCARD(savedd) without TO(outdd), you can further process the savedd records **after** SUBSET processing using one (and only one) OUTFIL statement like this:

    ```
    OUTFIL FNAMES=savedd,...
    ```

  - If you specify TO(outdd) and DISCARD(savedd), you can further process the outdd and savedd records **after** SUBSET processing using two (and only two) OUTFIL statements like this:

    ```
    OUTFIL FNAMES=outdd,...
    ```

    or multiple OUTFIL statements like this:

    ```
    OUTFIL FNAMES=outdd,...
    OUTFIL FNAMES=savedd,...
    ```

    Both statements must be specified in the order shown with at least the FNAMES parameter. For example, to further modify only the DISCARD data set, you could use statements like this:

    ```
    OUTFIL FNAMES=OUT
    OUTFIL FNAMES=SAVE,OMIT=(21,3,ZD,GT,+25)
    ```

- **VSAMTYPE(x)**

Specifies the record type for a VSAM input data set. x must be either F for fixed-length record processing or V for variable-length record processing.

If VSAMTYPE(x) is specified, ICETOOL will pass a RECORD TYPE=x control statement to DFSORT. (If you specify a RECORD TYPE=x statement in the xxxxCNTL data set, it will override the one passed by ICETOOL.)

Example 1

```
SUBSET FROM(IN1) TO(OUT1) REMOVE INPUT HEADER TRAILER
```

This example illustrates how you can remove the header record (first record) and trailer record (last record).

The VB input records might look like this:

```
Len|Data
 33|01A   RODENTS           FFFFF
 23|01A   VOLE    BINKY
 26|02B   HAMSTER GARFIELD
 22|03A   RAT     JUNE
 24|04B   MOUSE   MICKEY
 21|01A   COUNT   004
```

We just want to keep the data records. We use REMOVE, INPUT, HEADER and TRAILER to indicate we want to remove the header and trailer input records.

The output records look like this:

```
 23|01A   VOLE    BINKY
 26|02B   HAMSTER GARFIELD
 22|03A   RAT     JUNE
 24|04B   MOUSE   MICKEY
```

Example 2

```
SUBSET FROM(IN2) TO(OUT2) DISCARD(OUT3) -
  KEEP INPUT RRN(3,4) LAST(3)
```

This example illustrates how you can create one output file with relative records and the last n records, and another output file with the remaining records.

The input records might look like this:

```
Algebra
Astronomy
Biology
Calculus
French
Geography
Geometry
Greek
History
Latin
Psychology
Russian
```

In the first output file (OUT2), we want the third and fourth input records and the last three input records. In the second output file (OUT3), we want the records that are not in the first output file. We use TO(OUT2) and

DISCARD(OUT3) for the two output files. We use KEEP, INPUT, RRN(3,4) and LAST(3) to indicate we want to keep relative input records 3 and 4 and the last 3 input records.

The OUT1 records look like this:

```
Biology
Calculus
Latin
Psychology
Russian
```

The OUT2 records look like this:

```
Algebra
Astronomy
French
Geography
Geometry
Greek
History
```

Example 3

```
SUBSET FROM(IN3) TO(OUT4) KEEP OUTPUT -
  LAST(5) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,15,CH,A)
```

This example illustrates how you can keep the last 5 sorted records.

The input records might look like this:

```
Psychology
Biology
Russian
French
History
Geography
Calculus
Geometry
Algebra
Greek
Astronomy
Latin
```

We want to sort the records by the CH field in positions 1-15 and keep the last 5 sorted records. We use KEEP, OUTPUT and LAST(5) to keep the last 5 output records. We use the SORT statement to sort the records before they are output. After the SORT statement is executed, the intermediate records look like this:

```
Algebra
Astronomy
Biology
Calculus
French
Geography
Geometry
Greek
History
Latin
Psychology
Russian
```

After the SUBSET operator is executed, the final output records look like this:

```
Greek
History
Latin
Psychology
Russian
```

SELECT with first n duplicates

Introduction

ICETOOL's SELECT operator now allows you to select the first n records with each key or the first n duplicate records with each key. This gives you new capabilities for selecting records representing "top" and "bottom" categories.

As an example, the following SELECT operator could be used to list the top 3 students in each class.

```
...
//TOOLIN DD *
SELECT FROM(IN) TO(OUT) ON(1,10,CH) FIRST(3) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,10,CH,A,21,2,ZD,D)
```

We sort on the class name in ascending order, and on the average in descending order, to get the records in order by class name and highest to lowest average. We SELECT on the class name and use FIRST(3) to get the first three records for each class which gives us up to three students with the highest average.

If the IN data set contained these records:

```
Geometry  Fred      85
Geometry  Janis     71
Geometry  Leonard   78
Geometry  Michael   91
Geometry  Susan     83
Geometry  William   92
Algebra   Fred      83
Algebra   Janis     90
Algebra   Leonard   85
Algebra   Michael   94
Algebra   Susan     92
Algebra   William   87
```

the OUT data set would contain these records:

```
Algebra   Michael   94
Algebra   Susan     92
Algebra   Janis     90
Geometry  William   92
Geometry  Michael   91
Geometry  Fred      85
```

Syntax

The syntax for the new operands of SELECT is as follows:

```
SELECT ... FIRST(x)|FIRSTDUP(x)
```

Detailed Description

The new FIRST(x) operand of SELECT keeps the first x records with each unique key. The new FIRSTDUP(x) operand of SELECT keeps the first x records for each set of duplicate values.

FIRST(x) or FIRSTDUP(x) can be used to specify the criteria for selecting records in the same way the existing operands ALLDUPS, NODUPS, HIGHER(x), LOWER(x), EQUAL(x), FIRST, LAST, FIRSTDUP or LASTDUP can be used. One (and only one) of these operands must be specified.

The new operands of SELECT described below, and the existing operands of SELECT, can be specified in any order:

- **FIRST(x)**

  Limits the records selected to those with ON values that occur only once (value count = 1) and the first x records of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the first x records for each unique field value. For example, FIRST(3) keeps the first 3 records for each key.

  x must be specified as n or +n where n can be 1 to 999999999999999.

  DFSORT symbols can be used for n and +n in FIRST(x).

  FIRST(1) is equivalent to FIRST.

- **FIRSTDUP(x)**

  Limits the records selected to the first x records of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the first x records of those records with duplicate field values. For example, FIRSTDUP(5) keeps the first 5 records for each duplicate key.

  x must be specified as n or +n where n can be 1 to 999999999999999.

  DFSORT symbols can be used for n and +n in FIRSTDUP(x).

  FIRSTDUP(1) is equivalent to FIRSTDUP.

Example 1

```
SELECT FROM(INPUT) TO(HIGH) ON(8,3,CH) FIRST(2) USING(CTL1)
SELECT FROM(INPUT) TO(LOW)  ON(8,3,CH) FIRST(2) USING(CTL2)
//CTL1CNTL DD *
  SORT FIELDS=(8,3,CH,A,17,10,UFF,D)
//CTL2CNTL DD *
  SORT FIELDS=(8,3,CH,A,17,10,UFF,A)
```

This example illustrates how you can create one output file with records having the highest two values for each occurrence of a field, and another output file with records having the lowest two values for each occurrence of a field.

The input records might look like this:

```
BRANCH 001 JAN    5,231.87
BRANCH 001 FEB    1,983.21
BRANCH 001 MAR    2,103.52
BRANCH 001 APR      586.12
BRANCH 001 MAY   12,862.05
BRANCH 001 JUN    7,213.96
BRANCH 002 JAN      105.12
BRANCH 002 FEB    9,032.05
BRANCH 002 MAR    8,721.35
BRANCH 002 APR   14,635.32
BRANCH 002 MAY      936.28
BRANCH 002 JUN    6,012.38
```

In the first output file (HIGH), we want the records for each BRANCH for the two months with the highest amounts. In the second output file (LOW), we want the records for each BRANCH for the two months with the lowest amounts.

For the first SELECT operator to get the highest amounts, we use ON(8,3,CH) for the BRANCH code (001 or 002), FIRST(2) to get the first two records for each branch and a SORT statement that sorts on the branch ascending and the **amount descending**. After the SORT statement is executed, the intermediate records look like this:

```
BRANCH 001 MAY   12,862.05
BRANCH 001 JUN    7,213.96
BRANCH 001 JAN    5,231.87
BRANCH 001 MAR    2,103.52
BRANCH 001 FEB    1,983.21
BRANCH 001 APR      586.12
BRANCH 002 APR   14,635.32
BRANCH 002 FEB    9,032.05
BRANCH 002 MAR    8,721.35
BRANCH 002 JUN    6,012.38
BRANCH 002 MAY      936.28
BRANCH 002 JAN      105.12
```

After SELECT with FIRST(2) is executed, the final HIGH output looks like this:

```
BRANCH 001 MAY   12,862.05
BRANCH 001 JUN    7,213.96
BRANCH 002 APR   14,635.32
BRANCH 002 FEB    9,032.05
```

For the second SELECT operator to get the lowest amounts, we use ON(8,3,CH) for the BRANCH code (001 or 002), FIRST(2) to get the first two records for each branch and a SORT statement that sorts on the branch ascending and the **amount ascending**. After the SORT statement is executed, the intermediate records look like this:

```
BRANCH 001 APR      586.12
BRANCH 001 FEB    1,983.21
BRANCH 001 MAR    2,103.52
BRANCH 001 JAN    5,231.87
BRANCH 001 JUN    7,213.96
BRANCH 001 MAY   12,862.05
BRANCH 002 JAN      105.12
BRANCH 002 MAY      936.28
BRANCH 002 JUN    6,012.38
BRANCH 002 MAR    8,721.35
BRANCH 002 FEB    9,032.05
BRANCH 002 APR   14,635.32
```

After SELECT with FIRST(2) is executed, the final LOW output looks like this:

```
BRANCH 001 APR      586.12
BRANCH 001 FEB    1,983.21
BRANCH 002 JAN      105.12
BRANCH 002 MAY      936.28
```

Example 2

```
SELECT FROM(INPUT) TO(OUTPUT) ON(30,7,CH) FIRSTDUP(3)
```

This example illustrates how you can take a sample of up to 3 records with each duplicate value (that is, each value that occurs more than once).

The input records might look like this:

```
08015 062753 CODE 004 USERID 8720398
08005 100305 CODE 005 USERID 6030201
08007 110208 CODE 006 USERID 6030201
08012 110208 CODE 004 USERID 6030201
08018 050927 CODE 004 USERID 6030201
08004 060201 CODE 003 USERID 7137883
08004 060203 CODE 003 USERID 7137883
08017 030013 CODE 006 USERID 5831225
08015 130509 CODE 005 USERID 3072173
08018 020005 CODE 003 USERID 3072173
08021 180317 CODE 005 USERID 3072173
08025 090357 CODE 004 USERID 3072173
08029 111242 CODE 003 USERID 3072173
```

We want to select 3 records for each userid that has more than one record. We use ON(30,7,CH) for the userid and FIRSTDUP(3) to get the first 3 userid records for duplicates.

The output records look like this:

```
08015 130509 CODE 005 USERID 3072173
08018 020005 CODE 003 USERID 3072173
08021 180317 CODE 005 USERID 3072173
08005 100305 CODE 005 USERID 6030201
08007 110208 CODE 006 USERID 6030201
08012 110208 CODE 004 USERID 6030201
08004 060201 CODE 003 USERID 7137883
08004 060203 CODE 003 USERID 7137883
```

Note that we selected up to 3 records for the duplicate userids (3072173, 6030201 and 7137883) and did not select any records for the non-duplicate userids (8720398 and 5831225).

SPLICE with non-blank fields

Introduction

ICETOOL's SPLICE operator now allows you to create a single record for each key by splicing the base record with every specified nonblank field from each overlay record. This gives you new capabilities for collecting information from multiple records with the same key. You can now do a splice involving duplicate records with non-consecutive or missing fields, something that could not be accomplished previously.

As an example, you could use the following SPLICE operator to create one record for each name containing all of the colors for that name from a "matrix".

```
SPLICE FROM(IN) TO(OUT) ON(1,10,CH) KEEPNODUPS -
  WITHANY WITH(11,10) WITH(21,10) WITH(31,10) WITH(41,10)
```

If the IN data set contained these records:

```
VICKY               RED
VICKY                         GREEN
FRANK     GREEN
FRANK                                     RED
FRANK                         PURPLE
FRANK               BLUE
LEONARD                                   GREEN
DAVE                YELLOW
DAVE      BLUE
DAVE                                      PURPLE
```

the OUT data set would contain these records:

```
DAVE      BLUE      YELLOW              PURPLE
FRANK     GREEN     BLUE      PURPLE    RED
LEONARD                                 GREEN
VICKY               RED       GREEN
```

Syntax

The syntax for the new operand of SPLICE is as follows:

```
SPLICE ... WITHANY
```

Detailed Description

The new WITHANY operand of SPLICE can be used to create one spliced record for each set of duplicates. The first duplicate is spliced with the nonblank values of each subsequent duplicate for specified fields.

WITHANY can be used to specify how the records are to be spliced in the same way that WITHEACH or WITHALL can be used. These operands are optional and mutually exclusive.

The new operand of SPLICE described below, and the existing operands of SPLICE, can be specified in any order:

- **WITHANY**

  Specifies that the first duplicate, as defined by the ON fields, is spliced with each nonblank specified WITH field from each subsequent duplicate.

WITHANY overrides the default of splicing the first duplicate with all of the specified fields from the last duplicate.

The first duplicate is treated as a base record. Each subsequent duplicate is treated as an overlay record. Each specified field with a nonblank value in each overlay record is overlaid on to the base record. Thus, the output record consists of fields from the base record intermixed with specified nonblank fields from each overlay record. The value from the last overlay record with each nonblank value will appear in the output record. Note that a specified "field" from an overlay record can actually consist of multiple fields from the record that have previously been reformatted into one contiguous field.

The records to be spliced can originate from multiple input data sets.

To illustrate the splicing process when WITHANY is specified, if we had the following four fixed-length records with the base fields, ON field and WITH fields as shown:

```
BASE1  ON1     BASE2
       ON1                     WITHA
       ON1                           WITHB
       ON1            WITHC
```

The resulting spliced output record would be:

```
BASE1  ON1     BASE2  WITHC   WITHA  WITHB
```

For variable-length records, by default (without VLENMAX), the spliced record has the same length as the base record. For example, with WITHANY, if we had the following four records with the lengths (in the RDW), ON field and WITH fields as shown:

```
30 | BASE1 ON1           BASE2
50 |       ON1                                WITHB
25 |       ON1     WITHA
40 |       ON1                    WITHC
```

the resulting spliced output records would be:

```
30 | BASE1 ON1     WITHA BASE2
```

The WITHB and WITHC fields are beyond the end of the base record, so they are not spliced. However, if you specify VLENMAX, the spliced record is given the largest of the base record length or overlay record lengths. If the largest overlay record length is larger than the base record length, bytes in the extended spliced record that are not overlaid are filled in with blanks. The resulting spliced output record with WITHANY and VLENMAX would be:

```
50 | BASE1 ON1     WITHA BASE2   WITHC       WITHB
```

VLENOVLY cannot be specified with WITHANY.

Example 1

```
SPLICE FROM(IN) TO(OUT) ON(1,3,CH) WITHANY KEEPNODUPS -
  WITH(5,3) WITH(9,3) WITH(13,3) USING(CTL1)
//CTL1CNTL DD *
 INREC IFTHEN=(WHEN=(5,1,CH,EQ,C'1'),
           BUILD=(1,3,5:8,3)),
    IFTHEN=(WHEN=(5,1,CH,EQ,C'2'),
           BUILD=(1,3,9:8,3)),
    IFTHEN=(WHEN=(5,1,CH,EQ,C'3'),
           BUILD=(1,3,13:8,3))
```

This example illustrates how you can combine multiple rows for different types of data with a common key into a single row of data for that key, even if some rows are missing.

The input records might look like this:

```
001 2  150
001 3  120
001 1  100
002 3  140
002 1  250
003 1  050
003 2  920
004 3  005
```

We have an id number (for example, '001') in positions 1-3, a record type (1, 2 or 3) in position 5 and a numeric value in positions 8-10. We want to set up a single row for each id number with the values for the three records types and blanks for missing record types.

We use an INREC statement to move the value for each record to its position in the single row based on its record type. After the INREC statement is executed, the intermediate records look like this:

```
001     150
001         120
001 100
002         140
002 250
003 050
003     920
004         005
```

We use SPLICE with WITHANY and appropriate WITH fields to create one combined record for each id number with the values for the record types. We use KEEPNODUPS to keep records for id numbers with only one value (for example, '004').

The output records look like this:

```
001 100 150 120
002 250     140
003 050 920
004         005
```

DISPLAY with count

Introduction

ICETOOL's DISPLAY operator now allows you to display counts in reports. This gives you new capablities for printing break record count statistics and overall record count statistics in various forms in your reports.

As an example, you could use the following DISPLAY operator to list the number of pet rats, hamsters and gerbils, and the total number of adorable rodents.

```
DISPLAY FROM(IN) LIST(RPT) BLANK -
  BTITLE('Type of adorable pet: ') BREAK(1,10,CH) -
  BCOUNT('Adorable pets of this type: ') EDBCOUNT(U02) -
  HEADER('Pet''s name') ON(11,12,CH) -
  COUNT('Total number of adorable pets: ') EDCOUNT(U03)
```

If the IN data set contained these records:

```
Rat       Betty
Rat       Daisy
Rat       April
Rat       June
Rat       Buffy
Rat       Willow
Hamster   Spunky
Hamster   Baby
Gerbil    Willy
Gerbil    Binky
Gerbil    Marmaduke
```

the RPT data set would contain this report:

```
1Type of adorable pet:   Rat

 Pet's name
 ------------
 Betty
 Daisy
 April
 June
 Buffy
 Willow

 Adorable pets of this type:   6
1Type of adorable pet:   Hamster

 Pet's name
 ------------
 Spunky
 Baby

 Adorable pets of this type:   2
1Type of adorable pet:   Gerbil

 Pet's name
 ------------
 Willy
 Binky
 Marmaduke

 Adorable pets of this type:   3
1

 Total number of adorable pets:   11
```

Syntax

The syntax for the new operands of DISPLAY is as follows:

```
DISPLAY ... COUNT('string') EDCOUNT(formatting)
            BCOUNT('sting') EDBCOUNT(formatting)
```

Detailed Description

The new COUNT('string') and EDCOUNT(formatting) operands of DISPLAY specify printing of an overall count line in the list data set and indicate the text and formatting for the count line. The new BCOUNT('string') and EDBCOUNT(formatting) operands of DISPLAY specify printing of break count lines in the list data set and indicate the text and formatting for the count lines.

COUNT('string') can be used to print an overall count line in the same way the existing operands TOTAL('string'), MAXIMUM('string'), MINIMUM('string') and AVERAGE('string') can be used to print other overall statistics. EDCOUNT(formatting) can be used to edit the overall count. EDCOUNT(formatting) can only be specified if COUNT('string') is specified.

BCOUNT('string') can be used to print break count lines in the same way the existing operands BTOTAL('string'), BMAXIMUM('string'), BMINIMUM('string') and BAVERAGE('string') can be used to print other break statistics. BCOUNT('string') can only be specified if BREAK(p,m,f) or BREAK(p,m,f,formatting) is specified. EDBCOUNT(formatting) can only be specified if BCOUNT('string') is specified.

The new operands of DISPLAY described below, and the existing operands of DISPLAY, can be specified in any order:

- **COUNT('string')**

  Specifies an overall COUNT line is to be printed after the rows of data for the report. The specified string is printed starting at the indent column of the overall COUNT line, followed by the overall count of data records. If STATLEFT is specified, the string is printed to the left of the first column of data. If STATLEFT is not specified, the string is printed in the first column of data. The count is printed on the same line as the string. A blank line is printed before the overall COUNT line.

  The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('). To suppress printing of a string, specify COUNT('') using two single apostrophes.

  The count is printed in the format (PLUS, BLANK, or standard) you specify. EDCOUNT(formatting) can be used to apply formatting items to the count. The default number of digits (d) for the count is 15.

  The TOTAL, MAXIMUM, MINIMUM, AVERAGE and COUNT lines are printed in the order in which you specify them.

  DFSORT symbols can be used for 'string' in COUNT('string').

- **EDCOUNT(formatting)**

  Specifies how the overall count is to be formatted for printing. The BLANK operand is automatically in effect.

  The mask, L'string', F'string', T'string' and LZ formatting items can be used in the same way they are used for ON(p,m,f,formatting) as discussed in *"z/OS DFSORT Application Programming Guide"*.

  E'pattern' and Udd can be used as follows:

  E'pattern' specifies an edit pattern to be applied to the count. The pattern (1 to 24 characters) must be enclosed in single apostrophes. Each 9 in the pattern (up to 15) is replaced by a corresponding digit from the count. Characters other than 9 in the pattern appear as specified. To include a single apostrophe (') in the pattern, specify two single apostrophes ('). F'string' or a mask cannot be specified with E'pattern'.

  When E'pattern' is specified for the count:

  – If the number of significant digits in the count is less than the number of 9's in the pattern, 0's are filled in on the left. For example, 1234 is shown as 001234 with EDCOUNT(E'999999').

– If the number of significant digits in the count is greater than the number of 9's in the pattern, digits are truncated from the left. For example, 1234567 is shown as *4567* with EDCOUNT(E'*9999*').

Udd specifies the number of digits to be used for the count. dd specifies the number of digits and must be a two-digit number between 01 and 15. The default number of digits (d) for the count is 15.

If you know that your count requires less than 15 digits, you can use a lower number of digits (dd) instead by specifying Udd. For example, if EDCOUNT(U09) is specified, 9 digits (from U09) is used instead of 15 (default for the count).

If you use Udd and the count overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for Udd. For example, if EDCOUNT(U05) results in overflow, you can use EDCOUNT(U06) instead.

If E'pattern' is specified, Udd is ignored, because d is determined from the pattern.

- **BCOUNT('string')**

  Specifies a break COUNT line is to be printed after the rows of data for each section. The specified string is printed starting at the indent column of the break COUNT line, followed by the break count of data records in the section. If STATLEFT is specified, the string is printed to the left of the first column of data. If STATLEFT is not specified, the string is printed in the first column of data. The count is printed on the same line as the string. A blank line is printed before the break COUNT line.

  The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('). To suppress printing of a string, specify BCOUNT('') using two single apostrophes.

  The count is printed in the format (PLUS, BLANK, or standard) you specify. EDBCOUNT(formatting) can be used to apply formatting items to the count. The default number of digits (d) for the count is 15.

  The BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE and BCOUNT lines are printed in the order in which you specify them.

  DFSORT symbols can be used for 'string' in BCOUNT('string').

- **EDBCOUNT(formatting)**

  Specifies how the break count is to be formatted for printing. The BLANK operand is automatically in effect.

  See EDCOUNT(formatting) above for details on the formatting items you can use with EDBCOUNT(formatting).

Example 1

```
DISPLAY FROM(IN) LIST(RPT) -
  BTITLE('Division:') BREAK(1,10,CH) -
  HEADER('Branch Office') ON(11,15,CH) -
  HEADER('Profit/Loss (K)') ON(31,4,SFF,E1) -
  BMINIMUM('Lowest P/L in Division:') -
  BCOUNT('Offices in this Division: ') -
    EDBCOUNT(U02) -
  MINIMUM('Lowest P/L in Divisions:') -
  COUNT('Offices in all Divisions: ') -
    EDCOUNT(U02)
```

This example illustrates how you can produce a report with sections, containing a count for each section and a count for all of the sections.

The input records might look like this:

```
Chips     Gilroy              3293
Chips     Los Angeles         -141
Chips     Morgan Hill          213
Chips     Oakland             1067
Chips     San Francisco        -31
Chips     San Jose              92
Chips     San Martin          1535
Ice Cream Marin                673
Ice Cream Napa                  95
Ice Cream San Francisco       -321
Ice Cream San Jose            2318
Ice Cream San Martin            21
```

We use BCOUNT('string') to indicate we want a break count and the string we want to the left of the break count. We use EDBCOUNT(U02) to format the break count with 2 digits, overriding the default of 15 digits. We use COUNT('string') to indicate we want an overall count and the string we want to the left of the overall count. We use EDCOUNT(U02) to format the overall count with 2 digits, overriding the default of 15 digits.

The output report looks like this:

```
1Division:  Chips

 Branch Office     Profit/Loss (K)
 ---------------   ---------------
 Gilroy                     3,293
 Los Angeles                 (141)
 Morgan Hill                  213
 Oakland                    1,067
 San Francisco                (31)
 San Jose                      92
 San Martin                 1,535

 Lowest P/L in Division:     (141)

 Offices in this Division:   7
1Division:  Ice Cream

 Branch Office     Profit/Loss (K)
 ---------------   ---------------
 Marin                        673
 Napa                          95
 San Francisco               (321)
 San Jose                   2,318
 San Martin                   21

 Lowest P/L in Division:     (321)

 Offices in this Division:   5
1
 Branch Office     Profit/Loss (K)
 ---------------   ---------------

 Lowest P/L in Divisions:    (321)

 Offices in all Divisions:  12
```

DISPLAY/OCCUR with multiple and multipart titles

Introduction

ICETOOL's DISPLAY and OCCUR operators now allow you to display up to three title lines, each composed of up to three strings. This gives you new capabilities for printing multiline titles, and for using multiple strings for each title, including a combination of inline constants, and constants from DFSORT symbols including system information.

Other new options of DISPLAY and OCCUR also allow you to left justify the title lines instead of centering them, and to only display the title lines on the first page of the report instead of on every page of the report.

As an example, you could use the following OCCUR operator to print a report of possible system intruders with two lines for the title including system and location information:

```
...
//SYMNAMES DD *
System,S'&SYSNAME'
Sysplex,S'&SYSPLEX'
Location,'San Jose'
//TOOLIN DD *
  OCCUR FROM(FAILURES) LIST(CHECKIT) BLANK HIGHER(4) -
   DATE TITLE('Possible System Intruders on ',System) -
        TITLE(Sysplex,' in ',Location) PAGE -
        TBETWEEN(2) -
        HEADER(,'Userid') ON(23,8,CH) -
        HEADER('Logon failures','(More than 4)') ON(VALCNT)
```

The report in CHECKIT might look like this:

```
04/29/08  Possible System Intruders on EDS3  - 1 -

                  MAS3 in San Jose

             Logon failures
 Userid       (More than 4)
 --------   ---------------
 B7234510                 5
 D9853267                11
 ...
```

Syntax

The syntax for the changed operands of DISPLAY and OCCUR is as follows:

```
DISPLAY ... TITLE('string') ...
OCCUR       TITLE('string1','string2') ...
            TITLE('string1','string2','string3') ...
            TLEFT TFIRST
```

Detailed Description

The number of TITLE operands for DISPLAY and OCCUR has been expanded from one operand to up to three operands. Each operand can be used to create a title string on a separate line.

The TITLE operand of DISPLAY and OCCUR has been expanded from one string to up to three strings. It can now be specified as TITLE('string'), TITLE('string1','string2') or TITLE('string1','string2','string3'). The individual strings for a TITLE operand are combined to create one title string.

The new TLEFT operand of DISPLAY and OCCUR can be used to left justify the title lines, overriding the default of centering the title lines.

The new TFIRST operand of DISPLAY and OCCUR can be used to only print the title lines on the first page of the report, overriding the default of printing the title lines on every page of the report.

A single TITLE operand with a single string can be used to create one title string on one line. A single TITLE operand with two or three strings can be used to create one combined title string on one line. Up to two additional TITLE operands with one to three strings can be used to create title strings on multiple lines.

By default, the title lines are printed at the top of each page of the list data set. You can print title lines at the top of the first page only with TFIRST.

The first title line contains the first title string and the other title elements you specify (page number, date and time) in the order in which you specify them. The second title line contains the second title string. The third title line contains the third title string. A blank line is printed after each title line.

By default, the specified title strings are centered with respect to each other. You can left justify the title strings with TLEFT.

By default, eight blanks appear between title elements. You can change the space between title elements with TBETWEEN(n).

The new operands of DISPLAY and OCCUR described below, and the existing operands of DISPLAY and OCCUR, can be specified in any order:

- **TITLE('string')**

  Specifies a title string of the form:

  ```
  string
  ```

  The string can be up to 50 characters. It must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('').

  DFSORT symbols can be used for 'string' in TITLE('string').

- **TITLE('string1','string2')**

  Specifies a multipart title string of the form:

  ```
  string1string2
  ```

  The total combined length of string1 and string2 can be up to 50 characters. Each string must be enclosed in single apostrophes. To include a single apostrophe (') in a string, specify two single apostrophes ('').

  DFSORT symbols can be used for 'string1' and 'string2' in TITLE('string1','string2').

- **TITLE('string1','string2','string3')**

Specifies a multipart title string of the form:

```
string1string2string3
```

The total combined length of string1, string2 and string3 can be up to 50 characters. Each string must be enclosed in single apostrophes. To include a single apostrophe (') in a string, specify two single apostrophes ('').

DFSORT symbols can be used for 'string1', 'string2' and 'string3' in TITLE('string1','string2','string3').

- **TLEFT**

  Specifies that the title strings are to be left justified, overriding the default of centering the title strings with respect to each other.

- **TFIRST**

  Specifies that the title lines are only to appear on the first page of the report, overriding the default of having the title lines appear on every page of the report.

Example 1

```
DISPLAY FROM(VARDS) LIST(RDWLIST) -
  DATE(DMY.) -
  TFIRST TBETWEEN(3) -
  TITLE('Fancy RDW Report with') -
  TITLE('length in decimal and hex') -
  TITLE('and max and min length') -
  TIME(12:) -
  HEADER('Relative Record') ON(NUM) -
  HEADER('    RDW (length)') ON(VLEN) -
  HEADER('RDW (Hex)') ON(1,4,HEX) -
  BLANK -
  MINIMUM('Smallest Record:') -
  MAXIMUM('Largest Record:')
```

This example illustrates how you can produce a report with multiline titles.

The input data set is a VB file.

We use three TITLE operands to create a title with three lines. We use TFIRST to print the title lines at the top of the first page but not at the top of the other pages, overriding the default of printing the titles lines at the top of every page. By default, the title lines are centered (TLEFT would left align them).

The output report looks like this:

```
130.06.08     Fancy RDW Report with     05:37:43 pm

             length in decimal and hex

              and max and min length

 Relative Record       RDW (length)   RDW (Hex)
 ---------------   ----------------   ---------
               1                 84   00540000
               2                 47   002F0000
               3                 31   001F0000
               4                 31   001F0000
               5                 31   001F0000
 ...
1Relative Record       RDW (length)   RDW (Hex)
 ---------------   ----------------   ---------
              51                 31   001F0000
              52                 31   001F0000
              53                 31   001F0000
              54                 31   001F0000
              55                 31   001F0000
              56                 31   001F0000

 Smallest Record:                31

 Largest Record:                 84
```

DISPLAY/OCCUR without carriage control

Introduction

ICETOOL's DISPLAY and OCCUR operators now allow you to create reports without carriage control characters and with RECFM=FB instead of RECFM=FBA. This gives you new capabilities for suppressing the carriage control character. A blank line is used instead of a page eject control character to separate elements of the report.

As an example, you could use the following DISPLAY operator to list the number of pet rats, hamsters and gerbils, and the total number of adorable rodents, without carriage control characters (see "DISPLAY with count" to see what the report looks like with carriage control characters).

```
DISPLAY FROM(IN) LIST(RPT) BLANK NOCC -
  BTITLE('Type of adorable pet: ') BREAK(1,10,CH) -
  BCOUNT('Adorable pets of this type: ') EDBCOUNT(U02) -
  HEADER('Pet''s name') ON(11,12,CH) -
  COUNT('Total number of adorable pets: ') EDCOUNT(U03)
```

If the IN data set contained these records:

```
Rat       Betty
Rat       Daisy
Rat       April
Rat       June
Rat       Buffy
Rat       Willow
Hamster   Spunky
Hamster   Baby
Gerbil    Willy
Gerbil    Binky
Gerbil    Marmaduke
```

the RPT data set would contain this report:

```
Type of adorable pet:   Rat

Pet's name
------------
Betty
Daisy
April
June
Buffy
Willow

Adorable pets of this type:   6

Type of adorable pet:   Hamster

Pet's name
------------
Spunky
Baby

Adorable pets of this type:   2

Type of adorable pet:   Gerbil

Pet's name
------------
Willy
Binky
Marmaduke

Adorable pets of this type:   3



Total number of adorable pets:   11
```

Syntax

The syntax for the new operand of DISPLAY and OCCUR is as follows:

```
DISPLAY ... NOCC
OCCUR
```

Detailed Description

The new NOCC operand of DISPLAY and OCCUR specifies that carriage control characters are not to be used for the report.

The new operand of DISPLAY and OCCUR described below, and the existing operands of DISPLAY and OCCUR, can be specified in any order:

- **NOCC**

  Specifies that carriage control characters are not to be included in the lines of the list data set, overriding the default of using a carriage control character as the first byte of each line. A blank line is used instead of a page eject control character to separate elements of the report.

  The RECFM of the list data set is set to FB.

  The LRECL of the list data set will not include a byte for the carriage control character. If the line length is less than or equal to 120 bytes, the LRECL will be set to 120. If the line length is greater than 120 bytes, the LRECL will be set to the line length. The maximum line length is 2047 bytes.

  If the WIDTH(n) operand is specified, n can be 121 to 2047.

Example 1

```
OCCUR FROM(IN) LIST(VOLSERS) -
  NOCC -
  NOHEADER -
  ON(1,6,CH)
```

This example illustrates how you can produce a report listing unique volume serials with no carriage control characters and with RECFM=FB instead of the default of RECFM=FBA.

We use NOCC to indicate we don't want carriage control characters.

Example 2

```
DISPLAY FROM(IN) LIST(RPT) -
  NOCC -
  BTITLE('Division:') BREAK(1,10,CH) -
  HEADER('Branch Office') ON(11,15,CH) -
  HEADER('Profit/Loss (K)') ON(31,4,SFF,E1) -
  BMINIMUM('Lowest P/L in Division:') -
  BCOUNT('Offices in this Division: ') -
    EDBCOUNT(U02) -
  MINIMUM('Lowest P/L in Divisions:') -
  COUNT('Offices in all Divisions: ') -
    EDCOUNT(U02)
```

This example illustrates how you can produce a report with sections, but without carriage control characters. See Example 1 of the "Display with count" topic above for more details of this report. The only difference between that example and this one is that we've specified NOCC here to indicate we don't want carriage control characters. The output has RECFM=FB instead of RECFM=FBA and looks like this:

```
Division:  Chips

Branch Office     Profit/Loss (K)
--------------   ---------------
Gilroy                      3,293
Los Angeles                  (141)
Morgan Hill                   213
Oakland                     1,067
San Francisco                (31)
San Jose                      92
San Martin                  1,535

Lowest P/L in Division:     (141)

Offices in this Division:   7

Division:  Ice Cream

Branch Office     Profit/Loss (K)
--------------   ---------------
Marin                        673
Napa                          95
San Francisco               (321)
San Jose                    2,318
San Martin                    21

Lowest P/L in Division:     (321)

Offices in this Division:   5


Branch Office     Profit/Loss (K)
--------------   ---------------

Lowest P/L in Divisions:    (321)

Offices in all Divisions:  12
```

COUNT in output record

Introduction

ICETOOL's COUNT operator now allows you to create a count data set with an output record containing the record count. This gives you new capabilities for creating an output data set with a record containing text and the record count in various forms.

New options allow you to specify the ddname of the countdd data set, a text string to precede the count, and how the count should be formatted.

As an example, you could use the following COUNT operator to create one record with a 10-digit count:

```
COUNT FROM(IN) WRITE(CT) DIGITS(10)
```

If the IN data set contained 123,456 records, the CT data set would contain this 10-byte record:

```
0000123456
```

As another example, you could use the following COUNT operator to create one record with a text string and an edited count:

```
COUNT FROM(EMPIN) WRITE(EMPCT) -
  TEXT('Number of employees is ') -
  EDCOUNT(A1,U08) WIDTH(80)
```

If the EMPIN data set contained 1,234,567 records, the EMPCT data set would contain this 80-byte record:

```
Number of employees is   1,234,567
```

Syntax

The syntax for the new operands of COUNT is as follows:

```
COUNT ... WRITE(countdd) TEXT('string')
          DIGITS(d)|EDCOUNT(formatting) WIDTH(n)
```

Detailed Description

The new WRITE(countdd) operand of COUNT specifies the ddname of a count data set in which a count record will be written. By default, the count starts in the first byte of the count record and consists of 15 decimal digits with leading zeros. The new TEXT('string') operand can be used to specify a text string to precede the count. The new DIGITS(d) operand can be used to specify the number of digits for the count. The new EDCOUNT(formatting) operand can be used to format the count in various ways. The new WIDTH(n) operand can be used to set the LRECL of the count data set.

TEXT('string'), DIGITS(d), EDCOUNT(formatting) and WIDTH(n) can only be specified if WRITE(countdd) is specified. DIGITS(d) and EDCOUNT(formatting) are mutually exclusive.

The new operands of COUNT described below, and the existing operands of COUNT, can be specified in any order:

- **WRITE(countdd)**

  Specifies the ddname of the count data set to be produced by ICETOOL for this operation. A countdd DD statement must be present. ICETOOL sets the attributes of the count data set as follows:

  – RECFM is set to FB.

  – LRECL is set to one of the following:

    — If WIDTH(n) is specified, LRECL is set to n. Use WIDTH(n) if your count record length and LRECL must be set to a particular value (for example, 80), or if you want to ensure that the count record length does not exceed a specific maximum (for example, 20 bytes).

    — If WIDTH(n) is not specified, LRECL is set to the calculated required record length. If your LRECL does not need to be set to a particular value, you can let ICETOOL determine and set the appropriate LRECL value by not specifying WIDTH(n).

  – BLKSIZE is set to one of the following:

    — The BLKSIZE from the DD statement, DSCB, or label, if it is a multiple of the LRECL used.

    — The LRECL if the BLKSIZE from the DD statement, DSCB, or label is not a multiple of the LRECL used.

    — The system determined blocksize if the BLKSIZE is not available from the DD statement, DSCB, or label.

- **TEXT('string')**

Specifies a string to be printed starting in the first byte of the count record. The count follows the string.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('). To suppress printing of a string, do not specify TEXT('string') or specify TEXT('') using two single apostrophes.

DFSORT symbols can be used for 'string' in TEXT('string').

- **DIGITS(d)**

  Specifies d digits for the count, overriding the default of 15 digits. d can be 1 to 15.

  If you know that your count requires less than 15 digits, you can use a lower number of digits (d) instead by specifying DIGITS(d). For example, if DIGITS(10) is specified, 10 digits are used instead of 15.

  If you use DIGITS(d) and the count overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher d value for DIGITS(d). For example, if DIGITS(5) results in overflow, you can use DIGITS(6) instead.

- **EDCOUNT(formatting)**

  Specifies how the count is to be formatted for printing. The mask, L'string', F'string', T'string', LZ, E'pattern' and Udd formatting items can be used for the count as discussed for EDCOUNT(formatting) in "DISPLAY with count".

- **WIDTH(n)**

  Specifies the record length and LRECL you want ICETOOL to use for the count data set. n can be from 1 to 32760. ICETOOL always calculates the record length required to write the count record and uses it as follows:

  – If WIDTH(n) is specified and the calculated record length is less than or equal to n, ICETOOL sets the record length and LRECL to n. ICETOOL pads the count record on the right with blanks to the record length.

  – If WIDTH(n) is specified and the calculated record length is greater than n, ICETOOL issues an error message and terminates the operation.

  – If WIDTH(n) is not specified, ICETOOL sets the record length and LRECL to the calculated record length.

  Use WIDTH(n) if your count record length and LRECL must be set to a particular value (for example, 80), or if you want to ensure that the count record length does not exceed a specific maximum (for example, 20 bytes). Otherwise, you can let ICETOOL calculate and set the appropriate record length and LRECL by not specifying WIDTH(n).

Example 1

```
COUNT FROM(IN1) WRITE(CT1) DIGITS(6)
```

This example illustrates how you can write a count of the input records into an output record.

We use WRITE(CT1) to indicate the ddname of the output data set. We use DIGITS(6) to write the count as 6 digits, overriding the default of 15 digits.

If the input data set contains 8125 records, the 6-byte CT1 record would look like this:

```
008125
```

Example 2

```
COUNT FROM(IN2) WRITE(CT2) TEXT('Count is ') -
  EDCOUNT(A1,U10) WIDTH(80)
```

This example illustrates how you can write text and a formatted count of the input records into an output record of a specified length.

We use WRITE(CT2) to indicate the ddname of the output data set. We use TEXT('string') to write the string in the record before the count. We use EDCOUNT(A1,U10) to write the count as 10 digits with comma separators, overriding the default of 15 digits with no separators. We use WIDTH(80) to set the output record length to 80, overriding the default of the length of the string and count.

If the input data set contains 3286721 records, the 80-byte output record would look like this:

```
Count is      3,286,721
```

COUNT with add/subtract

Introduction

ICETOOL's COUNT operator now allows you to add a value to, or subtract a value from, the record count. This gives you new capabilities for increasing or decreasing the actual record count to get a resulting modified record count. This is especially useful for dealing with data sets that contain header and/or trailer records.

The resulting modified record count is displayed in the count message in TOOLMSG and in the count data set, and used to determine if the criteria specified for the count is satisfied (e.g. empty data set).

As an example, you could use this COUNT operator to write a record with the count of the data records between the header record and trailer record, and set a return code of 4 if the input data set only has a header record and a trailer record, but no data records.

```
COUNT FROM(IN) EMPTY RC4 SUB(2) WRITE(OUT) -
 TEXT('Number of data records is ')
```

If the IN data set contained these records:

```
20080425
Hammer     15
20080426
```

The OUT data set would contain:

```
Number of data records is 000000000000001
```

ICETOOL would set RC=0 for the COUNT operator since the EMPTY condition is not satisfied.

If the IN data set contained these records:

```
20080425
20080426
```

The OUT data set would contain:

```
Number of data records is 000000000000000
```

ICETOOL would set RC=4 for the COUNT operator since the EMPTY condition is satisfied.

Note that without SUB(2), ICETOOL would set RC=0 in both cases.

Syntax

The syntax for the new operands of COUNT is as follows:

```
COUNT ... ADD(x)|SUB(x)
```

Detailed Description

The new ADD(x) option of COUNT increases the record count by x. The new SUB(x) option of COUNT decreases the record count by x.

ADD(x) and SUB(x) are mutually exclusive.

The new operands of COUNT described below, and the existing operands of COUNT, can be specified in any order:

- **ADD(x)**

  Adds x to the record count. The resulting modified record count is displayed in the count message in TOOLMSG. If WRITE(countdd) is specified, the modified record count is used in the count record. If EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) is specified, the modified record count is used to determine if the criteria is satisfied. If ICETOOL was called using the parameter list interface, the "count of records processed" value in the Return Area for the COUNT operator is set to the modified record count.

  x must be specified as n or +n where n can be 1 to 999.

  DFSORT symbols can be used for n and +n in ADD(x).

- **SUB(x)**

  Subtracts x from the record count, but does not reduce the count below 0. The resulting modified record count is displayed in the count message in TOOLMSG. If WRITE(countdd) is specified, the modified record count is used in the count record. If EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) is specified, the modified record count is used to determine if the criteria is satisfied. If ICETOOL was called using the parameter list interface, the "count of records processed" value in the Return Area for the COUNT operator is set to the modified record count.

  x must be specified as n or +n where n can be 1 to 999.

  DFSORT symbols can be used for n and +n in SUB(x).

Example 1

```
COUNT FROM(IN3) WRITE(CT3) DIGITS(6) SUB(2)
```

This example illustrates how you can subtract 2 from the count of the input records and write the adjusted count into an output record.

We use WRITE(CT3) to indicate the ddname of the output data set. We use DIGITS(6) to write the count as 6 digits, overriding the default of 15 digits. We use SUB(2) to subtract 2 from the count of input records.

If the input data set contains 8125 records, the 6-byte CT3 record would look like this:

```
008123
```

SUB will not reduce the count below zero. For example, if the input data set contains 1 record, the 6-byte CT3 record would look like this:

```
000000
```

BLKSIZE default for input DUMMY

DFSORT will no longer terminate for a SORTIN DD DUMMY or SORTINnn DD DUMMY statement with RECFM and LRECL, but no BLKSIZE. Instead, DFSORT will use an appropriate BLKSIZE to process the DUMMY data set successfully.

Note: If DFSORT's Blockset technique is not selected, DFSORT may still terminate for a SORTIN DD DUMMY or SORTINnn DD DUMMY statement with RECFM and LRECL, but no BLKSIZE.

If the first SORTIN DD statement has DUMMY or DSN=NULLFILE with RECFM and LRECL attributes specified, but no BLKSIZE specified, DFSORT will process the data set using an appropriate BLKSIZE, for example, BLKSIZE=LRECL for RECFM=FB or BLKSIZE=LRECL+4 for RECFM=VB.

If any SORTINnn DD statement has DUMMY or DSN=NULLFILE with RECFM and LRECL attributes specified, but no BLKSIZE specified, DFSORT will process the data set using an appropriate BLKSIZE.

As an example, if this DD statement was specified for a DFSORT copy or sort operation:

```
//SORTIN DD DUMMY,RECFM=VB,LRECL=100
```

DFSORT would use BLKSIZE=104 to process the data set successfully.

As another example, if this DD statement was specified for a DFSORT merge operation:

```
//SORTIN02 DD DSN=NULLFILE,RECFM=FB,LRECL=2000
```

DFSORT would use BLKSIZE=2000 to process the data set successfully.

SKIP=0L default for SECTIONS

DFSORT will no longer terminate when an OUTFIL SECTIONS field is not followed by a SKIP, HEADER3 or TRAILER3 keyword. Instead, DFSORT will use a default keyword of SKIP=0L to process the sections successfully with no blank lines between sections for that field on the same page.

As an example, if this OUTFIL statement was specified:

```
OUTFIL SECTIONS=(11,4,21,8,HEADER3=('*******'))
```

DFSORT would process the OUTFIL statement above in the same way it would process this OUTFIL statement:

```
OUTFIL SECTIONS=(11,4,SKIP=0L,21,8,HEADER3=('*******'))
```

Thus, no blank lines will appear after each section associated with the 11,4 break field.

DFSORT symbols can be used for section fields as before.

SORTOUT=ddname default for FNAMES

DFSORT will now use the ddname specified by a SORTOUT=ddname operand in DFSPARM, the ddname specified by a SORTOUT=ddname operand in a parameter list, or the ddname specified in a TO(ddname) operand of an ICETOOL operator, as the default ddname for an OUTFIL statement without a FNAMES or FILES operand.

As an example, if the following was specified for ICETOOL:

```
...
//TOOLIN DD *
SELECT FROM(IN) TO(OUT1) ON(5,4,ZD) FIRSTDUP USING(CTL1)
/*
//CTL1CNTL DD *
  OUTFIL OMIT=(25,3,ZD,EQ,+150)
  OUTFIL FNAMES=OUT2,SAVE
/*
```

OUT1 would be used as the ddname associated with the first OUTFIL statement, and OUT2 would be used as the ddname associated with the second OUTFIL statement. That is, DFSORT would process the OUTFIL statements above in the same way it would process these OUTFIL statements:

```
  OUTFIL FNAMES=OUT1,OMIT=(25,3,ZD,EQ,+150)
  OUTFIL FNAMES=OUT2,SAVE
```

Changed Messages

This section shows existing messages that have been changed significantly for PTF UK90013. Refer to *z/OS DFSORT Messages, Codes and Diagnosis Guide* for general information on DFSORT messages.

ICE018A

ICE018A INVALID OR MISSING FORMAT

This message will be issued for the BEGIN and END operands of IFTHEN for the same reasons it is issued for the WHEN operand of IFTHEN.

ICE107A

ICE107A DUPLICATE, CONFLICTING, OR MISSING INREC OR OUTREC STATEMENT OPERAND

Explanation: Critical. One of the following errors was found in an INREC or OUTREC statement:

- An operand, other than IFTHEN, was specified twice. Example:

  ```
  INREC BUILD=(5,4,C'***',40:X),BUILD=(1,60)
  ```

- PARSE, FIELDS, BUILD, OVERLAY or FINDREP was specified with IFTHEN or IFOUTLEN. Example:

  ```
  OUTREC FINDREP=(IN=C'ONE',OUT=C'TWO'),
    IFTHEN=(WHEN=INIT,OVERLAY=(25:C'YES'))
  ```

- FIELDS and BUILD, FIELDS and OVERLAY, FIELDS and FINDREP, BUILD and OVERLAY, BUILD and FINDREP, or OVERLAY and FINDREP were specified. Example:

  ```
  OUTREC BUILD=(1,20),OVERLAY=(10:C'A')
  ```

- For an IFTHEN clause, WHEN was not specified. Example:

```
OUTREC IFTHEN=(OVERLAY=(10:C'A'))
```

- For an IFTHEN clause, WHEN=INIT, WHEN=(logexp), or WHEN=NONE was specified without PARSE, BUILD, OVERLAY or FINDREP. Example:

```
INREC IFTHEN=(WHEN=(5,1,CH,EQ,C'1'),HIT=NEXT)
```

- For an IFTHEN clause, WHEN=GROUP was specified without BEGIN, END or RECORDS, or without PUSH. Example:

```
INREC IFTHEN=(WHEN=GROUP,BEGIN=(9,2,CH,EQ,C'NO'))
```

- For an IFTHEN clause, WHEN=(logexp), WHEN=ANY, or WHEN=NONE was specified with PARSE, but without BUILD, OVERLAY or FINDREP. Example:

```
INREC IFTHEN=(WHEN=NONE,
      PARSE=(%01=(FIXLEN=5,ENDBEFR=BLANKS)))
```

- For an IFTHEN clause, WHEN=INIT, WHEN=(logexp), WHEN=ANY, or WHEN=NONE was specified with BEGIN, END, RECORDS or PUSH. Example:

```
OUTREC IFTHEN=(WHEN=INIT,PUSH=(9:5,8))
```

- For an IFTHEN clause, WHEN=GROUP was specified with PARSE, BUILD, OVERLAY or FINDREP. Example:

```
OUTREC IFTHEN=(WHEN=GROUP,BUILD=(9:5,8))
```

- An IFTHEN clause with WHEN=INIT was preceded by an IFTHEN clause with WHEN=(logexp), WHEN=ANY or WHEN=NONE. Example:

```
OUTREC IFTHEN=(WHEN=(5,2,CH,EQ,C'AA'),
                OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=INIT,BUILD=(1,80))
```

- An IFTHEN clause with WHEN=GROUP was preceded by an IFTHEN clause with WHEN=(logexp), WHEN=ANY or WHEN=NONE. Example:

```
OUTREC IFTHEN=(WHEN=(5,2,CH,EQ,C'AA'),
                OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=GROUP,RECORDS=3,PUSH=(8:SEQ=2)))
```

- An IFTHEN clause with WHEN=NONE was followed by an IFTHEN clause with WHEN=INIT, WHEN=(logexp), or WHEN=ANY. Example:

```
INREC IFTHEN=(WHEN=NONE,OVERLAY=(10:C'A')),
      IFTHEN=(WHEN=ANY,BUILD=(1,80))
```

- The first IFTHEN clause with WHEN=ANY was not preceded by an IFTHEN clause with WHEN=(logexp). Example:

```
OUTREC IFTHEN=(WHEN=INIT,OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=ANY,BUILD=(1,80))
```

- An IFTHEN clause with WHEN=ANY and without HIT=NEXT was followed by an IFTHEN clause with WHEN=ANY. Example:

```
OUTREC IFTHEN=(WHEN=(5,1,CH,EQ,C'1'),
         OVERLAY=(10:C'A'),HIT=NEXT),
       IFTHEN=(WHEN=(5,1,CH,EQ,C'2'),
         OVERLAY=(10:C'B'),HIT=NEXT),
       IFTHEN=(WHEN=ANY,
         OVERLAY=(28:C'ABC')),
       IFTHEN=(WHEN=ANY,BUILD=(1,80))
```

System Action: The program terminates.

Programmer Response: Check the INREC or OUTREC control statement for the errors indicated in the explanation and correct the errors.

ICE113A

ICE113A COMPARISON FIELD ERROR

This message will be issued for the BEGIN and END operands of IFTHEN for the same reasons it is issued for the WHEN operand of IFTHEN.

ICE114A

ICE114A INVALID COMPARISON

This message will be issued for the BEGIN and END operands of IFTHEN for the same reasons it is issued for the WHEN operand of IFTHEN.

ICE151A

ICE151A TOO MANY {*INCLUDE|*OMIT|*INREC|*OUTREC|ddname} IFTHEN n CONDITIONS

This message will be issued for the BEGIN and END operands of IFTHEN for the same reasons it is issued for the WHEN operand of IFTHEN.

ICE189A

ICE189A BLOCKSET REQUIRED BUT COULD NOT BE USED - REASON CODE IS nn

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

- FINDREP processing
- ICETOOL called DFSORT for DATASORT or SUBSET processing

ICE214A

ICE214A DUPLICATE, CONFLICTING, OR MISSING OUTFIL STATEMENT OPERANDS

Explanation: Critical. One of the following errors was found in an OUTFIL statement:

- An operand, other than IFTHEN, was specified twice. Example:

  ```
  OUTFIL STARTREC=5,STARTREC=10
  ```

- INCLUDE and OMIT, INCLUDE and SAVE, or OMIT and SAVE were specified. Example:

  ```
  OUTFIL INCLUDE=ALL,SAVE
  ```

- VTOF and CONVERT were specified. Example:

  ```
  OUTFIL VTOF,CONVERT
  ```

- FTOV and VTOF, FTOV and CONVERT, or FTOV and VLFILL were specified. Example:

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

- PARSE, OUTREC, BUILD, OVERLAY or FINDREP was specified with IFTHEN or IFOUTLEN. Example:

```
OUTFIL FINDREP=(IN=C'ONE',OUT=C'TWO'),
  IFTHEN=(WHEN=INIT,OVERLAY=(25:C'YES'))
```

- OUTREC and BUILD, OUTREC and OVERLAY, OUTREC and FINDREP, BUILD and OVERLAY, BUILD and FINDREP, or OVERLAY and FINDREP were specified. Example:

```
OUTFIL BUILD=(1,20),OVERLAY=(10:C'A')
```

- For an IFTHEN clause, WHEN was not specified. Example:

```
OUTFIL IFTHEN=(OVERLAY=(10:C'A'))
```

- For an IFTHEN clause, WHEN=INIT, WHEN=(logexp), or WHEN=NONE was specified without PARSE, BUILD, OVERLAY or FINDREP. Example:

```
OUTFIL IFTHEN=(WHEN=(5,1,CH,EQ,C'1'),HIT=NEXT)
```

- For an IFTHEN clause, WHEN=GROUP was specified without BEGIN, END or RECORDS, or without PUSH. Example:

```
OUTFIL IFTHEN=(WHEN=GROUP,BEGIN=(9,2,CH,EQ,C'NO'))
```

- For an IFTHEN clause, WHEN=(logexp), WHEN=ANY, or WHEN=NONE was specified with PARSE, but without BUILD, OVERLAY or FINDREP. Example:

```
OUTFIL IFTHEN=(WHEN=NONE,
      PARSE=(%01=(FIXLEN=5,ENDBEFR=BLANKS)))
```

- For an IFTHEN clause, WHEN=INIT, WHEN=(logexp), WHEN=ANY, or WHEN=NONE was specified with BEGIN, END, RECORDS or PUSH. Example:

```
OUTFIL IFTHEN=(WHEN=INIT,PUSH=(9:5,8))
```

- For an IFTHEN clause, WHEN=GROUP was specified with PARSE, BUILD, OVERLAY or FINDREP. Example:

```
OUTFIL IFTHEN=(WHEN=GROUP,BUILD=(9:5,8))
```

- For an IFTHEN clause, WHEN=INIT and BUILD with / were specified Example:

```
OUTFIL IFTHEN=(WHEN=INIT,BUILD=(1,25,/,26,25))
```

- For an IFTHEN clause, BUILD with / and HIT=NEXT were specified. Example:

```
OUTFIL IFTHEN=(WHEN=(21,1,CH,EQ,C'A'),
     BUILD=(1,25,/,26,25),HIT=NEXT)
```

- An IFTHEN clause with WHEN=INIT was preceded by an IFTHEN clause with WHEN=(logexp), WHEN=ANY or WHEN=NONE. Example:

```
OUTFIL IFTHEN=(WHEN=(5,2,CH,EQ,C'AA'),
               OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=INIT,BUILD=(1,80))
```

- An IFTHEN clause with WHEN=GROUP was preceded by an IFTHEN clause with WHEN=(logexp), WHEN=ANY or WHEN=NONE. Example:

```
OUTFIL IFTHEN=(WHEN=(5,2,CH,EQ,C'AA'),
               OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=GROUP,RECORDS=3,PUSH=(8:SEQ=2)))
```

- An IFTHEN clause with WHEN=NONE was followed by an IFTHEN clause with WHEN=INIT, WHEN=(logexp), or WHEN=ANY. Example:

```
OUTFIL IFTHEN=(WHEN=NONE,OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=ANY,BUILD=(1,80))
```

- The first IFTHEN clause with WHEN=ANY was not preceded by an IFTHEN clause with WHEN=(logexp). Example:

```
OUTFIL IFTHEN=(WHEN=INIT,OVERLAY=(10:C'A')),
       IFTHEN=(WHEN=ANY,BUILD=(1,80))
```

- An IFTHEN clause with WHEN=ANY and without HIT=NEXT was followed by an IFTHEN clause with WHEN=ANY. Example:

```
OUTFIL IFTHEN=(WHEN=(5,1,CH,EQ,C'1'),
         OVERLAY=(10:C'A'),HIT=NEXT),
       IFTHEN=(WHEN=(5,1,CH,EQ,C'2'),
         OVERLAY=(10:C'B'),HIT=NEXT),
       IFTHEN=(WHEN=ANY,
         OVERLAY=(28:C'ABC')),
       IFTHEN=(WHEN=ANY,BUILD=(1,80))
```

System Action: The program terminates.

Programmer Response: Check the OUTFIL control statement for the errors indicated in the explanation and correct the errors.

ICE221A

ICE221A INVALID FIELD OR CONSTANT IN {*INCLUDE|*OMIT|*INREC|*OUTREC|ddname} IFTHEN n CONDITION m

This message will be issued for the BEGIN and END operands of IFTHEN for the same reasons it is issued for the WHEN operand of IFTHEN.

ICE222A

ICE222A n BYTE FIXED RECORD LENGTH IS NOT EQUAL TO m BYTE LRECL FOR ddname

Explanation: Critical. The LRECL specified or retrieved for the fixed-length OUTFIL data set was not equal to the computed length of the output records for that data set. You cannot use the LRECL value to pad the OUTFIL records or to truncate the records produced by BUILD, OUTREC, OVERLAY, FINDREP, IFTHEN BUILD, IFTHEN OVERLAY, IFTHEN FINDREP or IFTHEN PUSH operand processing. The values shown in the message are as follows:

- n is the computed length of the output records for the OUTFIL group
- m is the specified or retrieved LRECL of the OUTFIL data set
- ddname indicates the OUTFIL data set for which padding or truncation was required

System Action: The program terminates.

Programmer Response: Take one of these actions as appropriate:

- Do not set the LRECL explicitly. Instead, let DFSORT set the LRECL to the computed record length.
- If you are using IFTHEN operands, specify IFOUTLEN=m. (Remember to allow an extra byte for OUTFIL report data sets for the ANSI carriage control character unless you specify the REMOVECC operand.)
- If you are not using IFTHEN operands, ensure that the computed length for the BUILD, OVERLAY or FINDREP operand, or the specified MAXLEN length for the FINDREP operand, is equal to m. (Remember to

allow an extra byte for OUTFIL report data sets for the ANSI carriage control character unless you specify the REMOVECC operand.)

ICE241A

ICE241A {*INREC|*OUTREC|ddname} IFTHEN n COLUMN OVERLAPS RECORD DESCRIPTOR WORD

Explanation: Critical. For variable-length record processing, the OVERLAY, IFTHEN OVERLAY or IFTHEN PUSH operand of an INREC, OUTREC or OUTFIL statement 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 specific cause of the error is identified as follows:

- *INREC and n=0 indicates that the OVERLAY operand of the INREC statement caused the error.
- *OUTREC and n=0 indicates that the OVERLAY operand of the OUTREC statement caused the error.
- ddname and n=0 indicates that the OVERLAY operand of an OUTFIL statement caused the error. ddname identifies the first data set in the associated OUTFIL group
- *INREC and n>0 indicates that an IFTHEN OVERLAY or IFTHEN PUSH operand of the INREC statement caused the error. n identifies the number of the associated IFTHEN clause (starting at 1 for the first IFTHEN clause in the INREC statement).
- *OUTREC and n>0 indicates that an IFTHEN OVERLAY or IFTHEN PUSH operand of the OUTREC statement caused the error. n identifies the number of the associated IFTHEN clause (starting at 1 for the first IFTHEN clause in the OUTREC statement).
- ddname and n>0 indicates that an IFTHEN OVERLAY or IFTHEN PUSH operand of an OUTFIL statement caused the error. ddname identifies the first data set in the associated OUTFIL group. n identifies the number of the associated IFTHEN clause (starting at 1 for the first IFTHEN clause in the OUTFIL statement).

The error is one of the following:

- c: was not specified for the first OVERLAY, IFTHEN OVERLAY or IFTHEN PUSH item so the default of 1: was used for that item. Example:

  ```
  OVERLAY=(C'ABC')
  ```

- c: was specified for an OVERLAY, IFTHEN OVERLAY or IFTHEN PUSH item with a value for c which was less than 5. Example:

  ```
  PUSH=(3:SEQ=5)
  ```

System Action: The program terminates.

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

```
OVERLAY=(8:C'ABC',1,2,HEX,25:5C'*')
```

ICE613A

ICE613A REQUIRED KEYWORD MISSING: keyword

Explanation: Critical. The indicated keyword was required for this operator, but was not specified. The required keywords and their operands for each operator are:

- COPY

  - FROM
  - TO or USING
- COUNT
  - FROM
  - EMPTY, NOTEMPTY, HIGHER, LOWER, EQUAL, or NOTEQUAL if RC4 is specified
  - WRITE if TEXT, DIGITS, EDCOUNT, or WIDTH is specified
- DATASORT
  - FROM, TO, and USING
  - HEADER or TRAILER
- DEFAULTS
  - LIST
- DISPLAY
  - FROM, ON, and LIST
  - BREAK if BTITLE, BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, or BCOUNT is specified
  - COUNT if EDCOUNT is specified
  - BCOUNT if EDBCOUNT is specified
- MODE
  - STOP, CONTINUE, or SCAN
- OCCUR
  - FROM and LIST
  - ON(p,m,f), ON(p,m,HEX), or ON(VLEN)
- RANGE
  - FROM and ON
  - HIGHER, LOWER, EQUAL, or NOTEQUAL
- SELECT
  - FROM and ON
  - TO or DISCARD
  - ALLDUPS, NODUPS, HIGHER, LOWER, EQUAL, FIRST, LAST, FIRSTDUP or LASTDUP
- SORT
  - FROM and USING
- SPLICE
  - FROM, TO, ON, and WITH
- STATS
  - FROM and ON
- SUBSET
  - FROM

  - TO or DISCARD
  - KEEP or REMOVE
  - INPUT or OUTPUT
- UNIQUE
  - FROM and ON
- VERIFY
  - FROM and ON

System Action: This operation is terminated.

Programmer Response: Supply the indicated keyword or operand.

ICE614A

ICE614A INVALID OPERATOR

Explanation: Critical. The first keyword in the statement was not a valid operator. The valid operators are: COPY, COUNT, DATASORT, DEFAULTS, DISPLAY, MODE, OCCUR (or OCCURS), RANGE, SELECT, SORT, SPLICE, STATS, SUBSET, UNIQUE, and VERIFY.

A common cause of this error is a missing hyphen (-) on the previous line to indicate continuation.

System Action: This operation is terminated.

Programmer Response:

A $ marks the point at which the error was detected. If an invalid operator was used, replace it with a valid operator. If this is a continuation line, use a hyphen after the last operand on the previous line.

ICE623A

ICE623A MAXIMUM NUMBER OF keyword KEYWORDS EXCEEDED

Explanation: Critical. Too many keywords of the indicated type were specified for this operator.

The maximum number of HEADER fields is 20 for a DISPLAY operator or 10 for an OCCUR operator.

The maximum number of WITH operands is 50 for a SPLICE operator.

The maximum number of TITLE operands is 3 for a DISPLAY or OCCUR operator.

The maximum number of RRN operands is 300 for a SUBSET operator.

The maximum number of ON fields for each operator is:

- DISPLAY - 20
- OCCUR - 10
- RANGE - 1
- SELECT - 10

- SPLICE - 10
- STATS - 10
- UNIQUE - 1
- VERIFY - 10

System Action: This operation is terminated.

Programmer Response: A $ marks the point at which the error was detected. Reduce the number of indicated keywords for this operator to the maximum allowed. If necessary, use additional operators to handle all the required fields.

ICE624A

ICE624A MAXIMUM NUMBER OF TO DDNAMES EXCEEDED

Explanation: Critical. Too many TO ddnames were specified for this operator. The maximum number of TO ddnames for each operator is:

- COPY - 10
- DATASORT - 1
- SELECT - 1
- SORT - 10
- SPLICE - 1
- SUBSET - 1

System Action: This operation is terminated.

Programmer Response: A $ marks the point at which the error was detected. Reduce the number of TO ddnames for this operator to the maximum allowed. Use additional operators to handle all the data sets required.

ICE628I

ICE628I RECORD COUNT: nnnnnnnnnnnnnnn

Explanation: Indicates the number of records processed by ICETOOL (prints as 15 decimal digits padded with zeros on the left as needed).

If ICETOOL completed the operation successfully, this count reflects the number of records in the input data set or in the subset of the input data set selected by DFSORT statements (for example, INCLUDE). If an ADD operand was specified for a COUNT operator, the count will reflect addition of the specified value. If a SUB operand was specified for a COUNT operator, the count will reflect subtraction of the specified value, but will not be reduced below 0.

If ICETOOL did not complete the operation successfully, this count reflects the number of records processed before an error was detected that caused ICETOOL to terminate processing of this operation.

System Action: None.

Programmer Response: None.

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 2048 or the maximum width specified. 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 2048 if WIDTH was not specified.

- For a DISPLAY or OCCUR operator with NOCC:

  The calculated record length for the indicated list data set was greater than 2047 or the maximum width specified. 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 2047 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. 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 m is less than 2048 without NOCC or less than 2047 with NOCC, either 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.

  If m is 2048 without NOCC or 2047 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.

ICE639A

ICE639A INSUFFICIENT MAIN STORAGE - ADD AT LEAST nK BYTES TO REGION

n has been increased from four digits to seven digits.

ICE640A

ICE640A INVALID FORMATTING ITEM

Explanation: Critical. An ON(p,m,f,formatting), ON(VLEN,formatting), ON(NUM,formatting), EDCOUNT(formatting) or EDBCOUNT(formatting) operand for this DISPLAY operator, or an ON(p,m,f,formatting), ON(VLEN,formatting) or ON(VALCNT,formatting) operand for this OCCUR operator, or an EDCOUNT operand for this COUNT operator, contained an invalid formatting item as follows:

- The formatting item was not /x (/x can be /D, /C, /K, /DK, /CK, /M, /G, /KB, /MB, or /GB), L'string', F'string', T'string', E'pattern', NOST, LZ, Ndd, Udd, or a valid mask (mask can be A0-A5, B1-B6, C1-C6, D1-D6, E1-E4, F1-F5, or G1-G6).
- /x, F'string', E'pattern', LZ, NOST, Ndd, Udd, or a mask was specified for a character field.
- /x or NOST was specified for ON(NUM,formatting) or for OCCUR.
- /x, NOST or Ndd was specified for BREAK(p,m,f,formatting), EDCOUNT(formatting) or EDBCOUNT(formatting).
- More than one /x was specified, more than one mask was specified, Ndd and Udd were both specified, or L'string', F'string', T'string', E'pattern', Udd, or Ndd was specified more than once.
- L'', F'', T'' or E'' was specified.
- F'string' or a mask was specified with E'pattern'.
- dd for Ndd or Udd was not two digits from 01 to 31.
- dd for Ndd or Udd was greater than 15 for ON(NUM,formatting) or ON(VALCNT,formatting).
- dd for Udd was greater than 15 for EDCOUNT(formatting) or EDBCOUNT(formatting).

System Action: The operation is terminated.

Programmer Response: A $ marks the point at which the error was detected. Correct the error.

ICE643I

ICE643I WIDTH OF REPORT IS n BYTES

Explanation: n indicates the line length and the LRECL for this DISPLAY or OCCUR list data set, determined as follows:

- n if WIDTH(n) was specified
- 121 if NOCC and WIDTH(n) were not specified and the calculated line length was less than or equal to 121
- the calculated line length if NOCC and WIDTH(n) were not specified and the calculated line length was greater than 121
- 120 if NOCC was specified, WIDTH(n) was not specified and the calculated line length was less than or equal to 120

- the calculated line length if NOCC was specified, WIDTH(n) was not specified and the calculated line length was greater than 120.

System Action: None.

Programmer Response: None.

ICE645A

ICE645A {(NUM)|(VALCNT)|COUNT} OVERFLOWED n DECIMAL DIGITS

Explanation: Critical. The number of digits (n) allowed from the Ndd, Udd or DIGITS(n) item you specified was too small, as follows:

- (NUM) indicates a record number for this DISPLAY operator exceeded the number of digits you allowed for it.
- (VALCNT) indicates a value count for this OCCUR operator exceeded the number of digits you allowed for it.
- COUNT indicates a count for this COUNT operator, or an overall count or break count for this DISPLAY operator, exceeded the number of digits you allowed for it.

System Action: This operation is terminated.

Programmer Response: Specify an Ndd or Udd formatting item, or a DIGITS(n) operand, large enough to prevent overflow of the record number, value count, overall count or break count (that is, use an appropriate Ndd, Udd or DIGITS value between n+1 and 15).

ICE652A

ICE652A OUTREC STATEMENT FOUND BUT NOT ALLOWED - USE OUTFIL STATEMENT INSTEAD

Explanation: Critical. A DFSORT OUTREC statement was specified for this DATASORT, SELECT, SPLICE or SUBSET operator, but you cannot use an OUTREC statement with DATASORT, SELECT, SPLICE or SUBSET.

System Action: The operation is terminated.

Programmer Response: If you want to reformat the output records produced by this DATASORT, SELECT, SPLICE or SUBSET operator, use one or more OUTFIL statements instead of an OUTREC statement.

New Messages

This section shows messages that have been added for PTF UK90013. Refer to *z/OS DFSORT Messages, Codes and Diagnosis Guide* for general information on DFSORT messages.

ICE259A

ICE259A PUSH FIELD ERROR

Explanation: Critical. The PUSH operand of an INREC, OUTREC or OUTFIL statement contained an invalid column, position or length, as follows:

- A 0 value was used.

- A column was greater than 32752, or was followed by another column.
- An input position plus length was greater than 32753.
- The length for a SEQ or ID field was greater than 15.
- An output field was beyond position 32767.

System Action: The program terminates.

Programmer Response: Correct the invalid column, position or length.

ICE260A

ICE260A FIND/REPLACE FIELD ERROR

Explanation: Critical. One of the following errors was found in the FINDREP operand of an INREC, OUTREC or OUTFIL statement:

- A 0 value was used.
- A null value was used where it was not permitted.
- IN or INOUT was not specified.
- INOUT, IN, OUT, STARTPOS, ENDPOS, DO, MAXLEN, OVERRUN or SHIFT was specified more than once.
- INOUT was specified with IN or OUT.
- IN was specified without OUT, or OUT was specified without IN.
- An input constant was specified without a matching output constant in INOUT.
- A repetition factor was 0 or greater than 256 for a character string or hexadecimal string.
- The total length of a single or repeated character or hexadecimal constant was greater than 256 bytes.
- An invalid digit or an odd number of digits was specified for a hexadecimal string.
- The value for STARTPOS, ENDPOS or MAXLEN was greater than 32752.
- The value for DO was greater than 1000.

System Action: The program terminates.

Programmer Response: Check the FINDREP operand for the errors indicated in the explanation and correct the errors.

ICE261A

ICE261A FIND AND REPLACE CAUSED OVERRUN OF n BYTE <*INREC/*OUTREC/ddname> OUTPUT RECORD END

Explanation: Critical. OVERRUN=ERROR was specified or defaulted for the FINDREP operand of an INREC, OUTREC or OUTFIL statement, and processing of the find/replace constants caused nonblank characters to overrun the end of the maximum of n bytes allowed for the output record. This can occur when an output constant is longer than an input constant (for example, INOUT=(C'A',C'XYZ')), or when the length of the record to be created by FINDREP is less than the length of the original record (for example, when the input record is 80 bytes and MAXLEN=50 is used to decrease the length of the record).

The specific source of the error is identified as follows:

- *INREC indicates that a FINDREP operand in the INREC statement caused the error.
- *OUTREC indicates that a FINDREP operand in the OUTREC statement caused the error.
- ddname indicates that a FINDREP operand in an OUTFIL statement caused the error; ddname identifies the first data set in the associated OUTFIL group.

System Action: The program terminates.

Programmer Response: If you want to truncate nonblank characters that overrun the end of the record, specify OVERRUN=TRUNC. If you do not want to truncate nonblank characters that overrun the end of the record, increase the length of the record using MAXLEN=m with a value for m equal to or greater than the maximum length record to be created by FINDREP.

ICE653A

ICE653A STOPAFT, SKIPREC OR COND (SUBSET) OR STOPAFT (DATASORT) NOT ALLOWED

Explanation: Critical. Either:

- a SUBSET operator was used and a DFSORT SKIPREC, STOPAFT or COND operand was specified, or
- a DATASORT operator was used and a DFSORT STOPAFT operand was specified.

You cannot use STOPAFT with SUBSET or DATASORT. You cannot use SKIPREC or COND with SUBSET.

System Action: This operation is terminated.

Programmer Response: If appropriate, use an OUTFIL statement with INCLUDE, OMIT, STARTREC, ENDREC or other operands to remove unwanted records after they are processed by SUBSET or DATASORT.

ICE654A

ICE654A SORT FUNCTION IS REQUIRED FOR DATASORT OPERATION

Explanation: Critical. A DFSORT SORT statement was not found for this DATASORT operator, but you must supply a SORT statement with DATASORT.

System Action: This operation is terminated.

Programmer Response: Specify a SORT statement in the xxxxCNTL data set corresponding to the USING(xxxx) operand for this DATASORT operator. Ensure that the SORT statement is not overridden by an OPTION COPY statement.

ICE655I

ICE655I COUNT RECORD WRITTEN IN ddname DATA SET - LENGTH IS n BYTES

Explanation: The record containing the count was written in the output data set with the indicated ddname. n indicates the record length and the LRECL for the output data set, determined as follows:

- n if WIDTH(n) was specified
- the calculated length of the count record if WIDTH(n) was not specified

System Action: None.

Programmer Response: None.

ICE656A

ICE656A DISCARD AND INPUT CANNOT BE USED WITH SORT FUNCTION FOR SUBSET OPERATION

Explanation: Critical. A DFSORT SORT statement was found for a SUBSET operator with DISCARD(savedd) and INPUT operands, but you cannot use a SORT statement with these SUBSET operands.

System Action: This operation is terminated.

Programmer Response: Remove the SORT statement or DISCARD(savedd) operand, or change the INPUT operand to an OUTPUT operand, as appropriate. Alternatively, use two SUBSET operators, one with REMOVE and INPUT operands, a SORT statement, and no DISCARD(savedd) operand, and the other with KEEP and INPUT operands, a SORT statement, and no DISCARD(savedd) operand.