User Guide for DFSORT PTFs UK90007 and UK90006

April, 2006

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 **UK90007** and DFSORT R14 PTF **UK90006**, which were first made available in **April, 2006**.

These PTFs provide important enhancements to DFSORT and DFSORT's ICETOOL for extracting variable position/length fields (e.g. CSV, delimited fields, keyword separated fields, etc) into fixed parsed fields (PARSE, %nn); justifying and squeezing data (JFY, SQZ); comparing and inserting past and future date constants (DATEn-r, DATEn+r); testing for numerics and non-numerics (NUM); displaying hexadecimal floating-point values as integers (FL); splitting files contiguously (SPLIT1R); suppressing page ejects in reports (BLKCCH1, BLKCCH2, BLKCCT1); using symbols for output columns (sym:); using system symbols (e.g. &SYSPLEX) in symbol constants (S'string'); reformatting records before selecting or splicing (INREC, SELECT, SPLICE); sorting and merging with larger PD and ZD fields; easier migration from other sort products, and more.

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

Contents

User Guide for DFSORT PTFs UK90007 and UK90006

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 **UK90007** and DFSORT Release 14 PTF **UK90006**, which were first made available in **April, 2006**, provide important enhancements to DFSORT and DFSORT's ICETOOL for extracting variable position/length fields (e.g. CSV, delimited fields, keyword separated fields, etc) into fixed parsed fields (PARSE, %nn); justifying and squeezing data (JFY, SQZ); comparing and inserting past and future date constants (DATEn-r, DATEn+r); testing for numerics and non-numerics (NUM); displaying hexadecimal floating-point values as integers (FL); splitting files contiguously (SPLIT1R); suppressing page ejects in reports (BLKCCH1, BLKCCH2, BLKCCT1); using symbols for output columns (sym:); using system symbols (e.g. &SYSPLEX) in symbol constants (S'string'); reformatting records before selecting or splicing (INREC, SELECT, SPLICE); sorting and merging with larger PD and ZD fields; easier migration from other sort products, and more.

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

You can 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 PTF UK90007 or DFSORT R14 PTF UK90006. The information in this paper will be included in the z/OS DFSORT books at their next major update (but not in the DFSORT R14 books).

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 Enhancements

INCLUDE and OMIT Enhancements

COND now allows you to compare date fields in various formats to past and future dates (relative to the date of your DFSORT run) using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r, DATEnP-r, Y'DATEn'+r and Y'DATEn'-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and

&DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

COND now allows you to test a field for numerics (field,EQ,NUM) or non-numerics (field,NE,NUM) in character (FS), zoned decimal (ZD) or packed decimal (PD) format.

COND now allows you to use &DATEn, &DATEn(c) and &DATEnP as aliases for DATEn, DATEn(c) and DATEnP, respectively.

INREC and OUTREC Enhancements

PARSE and IFTHEN PARSE are new options that allow you to extract variable position/length fields into fixed-length parsed fields defined as %nn fields. PARSE gives you powerful new capabilities for handling variable fields such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and many other types.

You can use various PARSE options to define the rules for extracting variable fields into up to one hundred %nn fixed-length parsed fields (%00-%99), and then use these %nn fields where you can use p,m fields in BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS. You can edit, convert, justify, squeeze, translate, and do arithmetic with %nn fields.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS now allow you to use a new JFY option to left-justify or right-justify the data in a field. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally for JFY, specific leading and trailing characters can be changed to blanks before justification begins, a leading string can be inserted, a trailing string can be inserted, and the output length can be changed.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS now allow you to use a new SQZ option to left-squeeze or right-squeeze the data in a field. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-squeezed field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally for SQZ, specific characters can be changed to blanks before squeezing begins, a leading string can be inserted, a trailing string can be inserted, a string can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank, blanks can be kept as is between paired apostrophes or paired quotes, and the output length can be changed.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS now allow you to insert past and future dates (relative to the date of your DFSORT run) into your records in various forms using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and &DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS now allow you to use FL format to convert 4-byte or 8-byte hexadecimal floating-point values to integer values.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS now allow you to use new TO=PDF and TO=PDC options to convert numeric values to PD values with F or C for the positive sign, respectively. The TO=PDC option is equivalent to the existing TO=PD option.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and FIELDS now allow you to use &DATEn, &DATEn(c), &DATEnP, &YDDD=(abc), &YDDDNS=(ab), &TIMEn, &TIMEn(c) and &TIMEnP as aliases for DATEn, DATEn(c), DATEnP, YDDD=(abc), YDDDNS=(ab), TIMEn, TIMEn(c) and TIMEnP, respectively.

IFTHEN WHEN now allows you to compare date fields in various formats to past and future dates (relative to the date of your DFSORT run) using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r, DATEnP-r, Y'DATEn'+r and Y'DATEn'-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and &DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

IFTHEN WHEN now allows you to test a field for numerics (field,EQ,NUM) or non-numerics (field,NE,NUM) in character (FS), zoned decimal (ZD) or packed decimal (PD) format.

IFTHEN WHEN now allows you to use &DATEn, &DATEn(c) and &DATEnP as aliases for DATEn, DATEn(c) and DATEnP, respectively.

OUTFIL Enhancements

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

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

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

SPLIT1R is a new option that allows you to write contiguous groups of records in one rotation among multiple output data sets. A specified number of records is written to each output data set and extra records are written to the last output data set.

PARSE and IFTHEN PARSE are new options that allow you to extract variable position/length fields into fixed-length parsed fields defined as %nn fields. PARSE gives you powerful new capabilities for handling variable fields such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and many other types.

You can use various PARSE options to define the rules for extracting variable fields into up to one hundred %nn fixed-length parsed fields (%00-%99), and then use these %nn fields where you can use p,m fields in BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC. You can edit, convert, justify, squeeze, translate, and do arithmetic with %nn fields.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC now allow you to use a new JFY option to left-justify or right-justify the data in a field. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally for JFY, specific leading and trailing characters can be changed to blanks before justification begins, a leading string can be inserted, a trailing string can be inserted, and the output length can be changed.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC now allow you to use a new SQZ option to left-squeeze or right-squeeze the data in a field. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-squeezed field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally for SQZ, specific characters can be changed to blanks before squeezing begins, a leading string can be inserted, a trailing string can be inserted, a string can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank, blanks can be kept as is between paired apostrophes or paired quotes, and the output length can be changed.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC now allow you to insert past and future dates (relative to the date of your DFSORT run) into your records in various forms using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and &DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

TRAILERx, BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC now allow you to use FL format to convert 4-byte or 8-byte hexadecimal floating-point values to integer values.

TRAILERx, HEADERx, BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC now allow you to use new TO=PDF and TO=PDC options to convert numeric values to PD values with F or C for the positive sign, respectively. The TO=PDC option is equivalent to the existing TO=PD option.

BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY and OUTREC now allow you to use &DATEn, &DATEn(c), &DATEnP, &YDDD=(abc), &YDDDNS=(ab), &TIMEn, &TIMEn(c) and &TIMEnP as aliases for DATEn, DATEn(c), DATEnP, YDDD=(abc), YDDDNS=(ab), TIMEn, TIMEn(c) and TIMEnP, respectively.

INCLUDE, OMIT and IFTHEN WHEN now allow you to compare date fields in various formats to past and future dates (relative to the date of your DFSORT run) using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r, DATEnP-r, Y'DATEn'+r and Y'DATEn'-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and &DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

INCLUDE, OMIT and IFTHEN WHEN now allow you to test a field for numerics (field,EQ,NUM) or non-numerics (field,NE,NUM) in character (FS), zoned decimal (ZD) or packed decimal (PD) format.

INCLUDE, OMIT and IFTHEN WHEN now allow you to use &DATEn, &DATEn(c) and &DATEnP as aliases for DATEn, DATEn(c) and DATEnP, respectively.

Symbol Enhancements

A symbol can now be used for a %nn parsed field. For example, if Account,%01 is defined in SYMNAMES, Account can be used for %01. A symbol for %nn can be used in DFSORT control statements where %nn can be used. A symbol for %nn results in substitution of %nn.

A symbol can now be used for an output column. For example, if Start_address,18 is defined in SYMNAMES, Start_address: can be used for 18:. symbol: can be used in DFSORT control statements where c: can be used. A symbol for p or p,m or p,m,f results in substitution of p: for symbol: (output column).

A symbol can now be used for a new system symbol string constant. symbol,S'string' can be used to define a string containing any combination of EBCDIC characters and system symbols you want to use to form a character string. For example, if whererun,S'&JOBNAME. on &SYSPLEX' is defined in SYMNAMES, whererun can be used for the resulting constant. You can use dynamic system symbols such as &JOBNAME, &DAY, and so on, system-

defined static system symbols such as &SYSNAME, &SYSPLEX, and so on, and installation-defined static system symbols specified by your installation in an IEASYMxx member of SYS1.PARMLIB.

A symbol for a system symbol string can be used in DFSORT and ICETOOL control statements where a symbol for a character string can be used. DFSORT will replace each system symbol in S'string' with its substitution text to create a character string in the format C'new_string'.

ICETOOL Enhancements

DISPLAY now allows you to use FL format to convert 4-byte or 8-byte hexadecimal floating-point values to integer values.

DISPLAY and OCCUR now allow you to use a new TBETWEEN(n) option to specify the number of blanks between title elements (title, page number, date, time).

SELECT and SPLICE now allow you to use an INREC statement to reformat your records before they are selected or spliced. All of the operands of the INREC statement (PARSE, BUILD, OVERLAY, IFTHEN, IFOUTLEN and FIELDS) are now available with SELECT and SPLICE.

SORT and MERGE Enhancements

The maximum length for a PD or ZD sort or merge field has been raised to 256.

SUM Enhancements

The maximum position for the end of a sum field has been raised to 32752.

Other Enhancements

DFSORT supports large physical sequential data sets for input, output and work data sets.

DSA can now be specified as a run-time option. This allows you to adjust the maximum amount of storage available to DFSORT for dynamic storage adjustment of individual Blockset sort applications when SIZE/MAINSIZE=MAX is in effect.

DFSORT now accepts and ignores zero values in the starting and ending address of the RECORD statement image in the 24-Bit Parameter List. You can set these addresses to zero if you don't want to pass a control statement to DFSORT using the third and fourth words of the parameter list .

Operational Changes that may Require User Action

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

New reserved words for symbols

The following are new DFSORT/ICETOOL reserved words (uppercase only, as shown), which are no longer allowed as symbols: DATE1..., DATE2..., DATE3..., PDC and PDF.

If you used these words as symbols, you must change them to other words, such as lowercase or mixed case forms (for example, Date1p, date3(/) or pdc).

FL Conversion

DFSORT's INREC, OUTREC and OUTFIL statements, and ICETOOL's DISPLAY operator, can now convert FL (hexadecimal floating-point) values to integer values, providing you are running in z/Architecture mode. If you use FL in INREC, OUTREC, OUTFIL or DISPLAY when running in ESA/390 mode, FL will be recognized and the error messages issued may be different than those issued previously when FL was not allowed in INREC, OUTREC, OUTFIL or DISPLAY.

If you want to use FL in INREC, OUTREC, OUTFIL or DISPLAY, you must be running in z/Architecture mode.

OUTREC Statement with SELECT or SPLICE

DFSORT will now issue error message ICE652A and terminate if an OUTREC statement is used with ICETOOL's SELECT or SPLICE operator. This will make it easier to identify this error condition.

Note: If ABEND is in effect, DFSORT will ABEND with U0652, but will not issue ICE652A.

If you want to reformat your records after SELECT or SPLICE processing, use an OUTFIL statement rather than an OUTREC statement.

Left-Justify and Right-Justify (JFY)

Introduction

The BUILD, OVERLAY, IFTHEN BUILD and IFTHEN OVERLAY operands of the INREC, OUTREC and OUTFIL statements now allow you to use a new JFY option to left-justify or right-justify the data in a field. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally for JFY, specific leading and trailing characters can be changed to blanks before justification begins, a leading string can be inserted, a trailing string can be inserted, and the output length can be changed.

Tutorial

Suppose you had input records that looked like this:

```
         History
  Psychology
                    Business
     Biology
         Computer Science
```

Since this is rather messy looking, you can use the following statements to left-justify the data to make it more presentable:

```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=LEFT))
```

1,30,JFY=(SHIFT=LEFT) indicates that you want to left-justify the data in positions 1-30. The results produced for this OUTREC statement are:

```
History
Psychology
Business
Biology
Computer Science
```

Alternatively, you can use the following statements to right-justify the data:

```
  OPTION COPY
  OUTREC FIELDS=(1,30,JFY=(SHIFT=RIGHT))
```

1,30,JFY=(SHIFT=RIGHT) indicates that you want to right-justify the data in positions 1-30. The results produced for this OUTREC statement are:

```
                       History
                    Psychology
                      Business
                       Biology
              Computer Science
```

DFSORT's justify feature also lets you add a leading string, a trailing string, or both, to the data. For example, you can use the following statements to surround the left-justified data with '<' and '>':

```
  OPTION COPY
  OUTREC FIELDS=(1,30,JFY=(SHIFT=LEFT,LEAD=C'<',TRAIL=C'>'))
```

The results produced for this OUTREC statement are:

```
<History>
<Psychology>
<Business>
<Biology>
<Computer Science>
```

LEAD=string specifies the leading string as a character or hexadecimal constant (1 to 50 bytes). TRAIL=string specifies the trailing string as a character or hexadecimal constant (1 to 50 bytes).

Notice that when you left-justify, the trailing string is placed directly to the right of the last non-blank character in your data.

If you want to right-justify instead of left-justify, you can use the following statements:

```
  OPTION COPY
  OUTREC FIELDS=(1,30,JFY=(SHIFT=RIGHT,LEAD=C'<',TRAIL=C'>'))
```

The results produced for this OUTREC statement are:

```
                     <History>
                  <Psychology>
                    <Business>
                     <Biology>
            <Computer Science>
```

Notice that when you right-justify, the leading string is placed directly to the left of the first non-blank character in your data.

If your leading or trailing string causes the output field to be longer than the input field, you will lose characters. In order to avoid that, you can increase the length of the output field with the LENGTH parameter. For example, suppose you had input records that looked like this:

```
   rats
 bats
  cats
```

and you added a leading string with these control statements:

```
  OPTION COPY
  OUTREC FIELDS=(1,7,JFY=(SHIFT=LEFT,LEAD=C'*I love ',TRAIL=C'*'))
```

Since your input field is 7 bytes, your output field is also 7 bytes by default, so your output fields are truncated to:

```
*I love
*I love
*I love
```

To avoid truncation, you can use LENGTH=16 to increase the output field length by the 9 characters you added for the leading and trailing strings:

```
  OPTION COPY
  OUTREC FIELDS=(1,7,JFY=(SHIFT=LEFT,LEAD=C'*I love ',TRAIL=C'*',
    LENGTH=16))
```

The larger output field can accomodate your leading and trailing strings so you can show your appreciation for these wonderful creatures with the following output:

```
*I love rats*
*I love bats*
*I love cats*
```

The justify feature can also be used to remove leading and trailing characters other than blanks from your data. Suppose you had input records that looked like this:

```
         (History)
 (Psychology)
                    (Business)
    (Biology)
        (Computer Science)
```

You can use the following statements to remove the leading '(' character and the trailing ')' character before you left-justify the data:

```
  OPTION COPY
  OUTREC FIELDS=(1,30,JFY=(SHIFT=LEFT,PREBLANK=C'()'))
```

PREBLANK=list specifies a list of characters you want to replace with blanks before DFSORT starts to justify the data. You can specify the list as a character or hexadecimal constant (1 to 10 bytes). Remember that each character in the list is independent of the other characters. For example, PREBLANK=C'*/' replaces each leading or trailing '*' character and '/' character with a blank before justify procssing begins (for example, leading and trailing character sequences of /*, //*, */, // and * are all replaced with blanks).

The results produced for this OUTREC statement are:

```
History
Psychology
Business
Biology
Computer Science
```

You could use the following statements to right-justify the data and replace the leading '(' character and trailing ')' character with a leading '<' character and a trailing '>' character:

```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=RIGHT,PREBLANK=C'()',
   LEAD=C'<',TRAIL=C'>'))
```

The results produced for this OUTREC statement are:

```
                  <History>
               <Psychology>
                 <Business>
                  <Biology>
          <Computer Science>
```

Detailed Description and Syntax

You can use p,m,justify in the BUILD and OVERLAY operands of the INREC, OUTREC and OUTFIL statements.

p,m,justify

specifies that a left-justified or right-justified input field is to appear in the reformatted record. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

- specific leading and trailing characters can be changed to blanks before justification begins
- a leading string can be inserted
- a trailing string can be inserted
- the output length can be changed (it's equal to the input length by default)

p,m specifies the starting position and length of the 1 to 32752 byte field to be justified. The field must not extend beyond position 32752.

justify specifies how the input field is to be justified for output.

JFY

```
>>-JFY=(-+-SHIFT=LEFT--+-+-----------+-+----------------+------>
         '-SHIFT=RIGHT-' '-,LENGTH=n-' '-,PREBLANK=list-'

>--+--------------+-+--------------+--)------------------------><
   '-,LEAD=string-' '-TRAIL=string-'
```

Note: For clarity, in the examples below, b is used to represent a blank in the input fields.

SHIFT=LEFT

specifies that a left-justified input field is to appear in the reformatted record. Leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For example, with:

```
1,15,JFY=(SHIFT=LEFT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
ABbCDbbEFbbbbbb
```

m is used for the output field length unless LENGTH=n is specified. If the left-justified input field is shorter than the output field length, blanks are inserted on the right. If the left-justified input field is longer than the output field length, characters are truncated on the right. You can use LENGTH=n to prevent padding or truncation.

SHIFT=RIGHT

specifies that a right-justified input field is to appear in the reformatted record. Trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. For example, with:

```
1,15,JFY=(SHIFT=RIGHT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
bbbbbbABbCDbbEF
```

m is used for the output field length unless LENGTH=n is specified. If the right-justified input field is shorter than the output field length, blanks are inserted on the left. If the right-justified input field is longer than the output field length, characters are truncated on the left. You can use LENGTH=n to prevent padding or truncation.

LENGTH=n

specifies the length of the justified output field. The value for n must be between 1 and 32752. If LENGTH=n is not specified, m (the input field length) is used for the length of the justified output field. You can use LENGTH=n to prevent padding or truncation.

PREBLANK=list

specifies a list of one or more characters to be changed to blanks **before** justify processing begins. Only leading and trailing characters are changed to blanks. Scanning from left to right, each nonblank character at the start of the input field that matches a character in the list is replaced by a blank character, until a nonblank character not in the list is found. Scanning from right to left, each nonblank character at the end of the input field that matches a character in the list will be replaced by a blank character, until a nonblank character not in the list is found.

list can be 1 to 10 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

Each character in the list is treated as one character to be preblanked. Thus PREBLANK=C'*' specifies an asterisk is to be preblanked, and PREBLANK=C',;$' specifies a comma, a semicolon and a dollar sign are each to be preblanked. For example, let's say we have an input field of:

```
**b<*ABbCD*bEF>*b**
```

If we specify:

```
1,19,JFY=(SHIFT=LEFT,PREBLANK=C'*')
```

each leading or trailing asterisk is changed to a blank before left-justify processing begins. So the output field is:

```
<*ABbCD*bEF>bbbbbbb
```

If we specify:

```
1,19,JFY=(SHIFT=RIGHT,PREBLANK=C'*<>')
```

each leading or trailing asterisk, less than sign and greater than sign is changed to a blank before right-justify processing begins. So the output field is:

```
bbbbbbbbbbABbCD*bEF
```

LEAD=string

Specifies a string to be inserted in the output field before the first nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,JFY=(SHIFT=RIGHT,LEAD=C'XYZ')
```

the output field is:

```
bbbXYZABCbE
```

When we add characters with LEAD=string, it's often necessary to specify LENGTH=n to avoid truncation. For example, let's say we have an input field of:

```
AbBbC
```

If we specify:

```
1,5,JFY=(SHIFT=LEFT,LEAD=C'XYZ')
```

the output field is:

```
XYZAb
```

Since the output field length is defaulted to the input field length of 5, the resulting 8 characters (XYZAbBbC) are truncated on the right to 5 characters (XYZAb) for output. If we instead specify:

```
1,5,JFY=(SHIFT=LEFT,LEAD=C'XYZ',LENGTH=8)
```

the output field is:

```
XYZAbBbC
```

LENGTH=8 increases the output field by the 3 LEAD characters and truncation is prevented.

TRAIL=string

Specifies a string to be inserted in the output field after the last nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,JFY=(SHIFT=LEFT,TRAIL=C'XYZ')
```

the output field is:

```
ABCbEXYZbbb
```

When we add characters with TRAIL=string, it's often necessary to specify LENGTH=n to avoid truncation. For example, let's say we have an input field of:

```
AbBbC
```

If we specify:

```
1,5,JFY=(SHIFT=RIGHT,TRAIL=C'XYZ')
```

the output field is:

```
bCXYZ
```

Since the output field length is defaulted to the input field length of 5, the resulting 8 characters (AbBbCXYZ) are truncated on the left to 5 characters (bCXYZ) for output. If we instead specify:

```
1,5,JFY=(SHIFT=RIGHT,TRAIL=C'XYZ',LENGTH=8)
```

the output field is:

```
AbBbCXYZ
```

LENGTH=8 increases the output field by the 3 TRAIL characters and truncation is prevented.

Sample Syntax:

```
OUTFIL BUILD=(5:16,20,JFY=(SHIFT=LEFT,PREBLANK=C'*',
   LEAD=C'<A>',TRAIL=C'</A>',LENGTH=22))
```

Example 1

```
INREC OVERLAY=(16:1,15,JFY=(SHIFT=LEFT))
SORT FIELDS=(16,15,CH,A)
OUTREC BUILD=(1,15)
```

This example illustrates how you can left-justify characters in an input field so they can be sorted without regard to the leading blanks.

The 15-byte input records might look like this:

```
   CARRIE
       VICKY
          FRANK
      SAM
DAVID
    MARTIN
```

Note that if we sort these records using just this control statement:

```
   SORT FIELDS=(1,15,CH,A)
```

the 15-byte output records are as follows:

```
          FRANK
       VICKY
      SAM
    MARTIN
   CARRIE
DAVID
```

Because of the different number of leading blanks in the input records, we don't get what we want. To fix that, while keeping the leading blanks in the original records, we use the JFY function of INREC to make a left-justified

copy of the 15-byte input field at positions 16-30, SORT on it and use OUTREC to remove the left-justified field. With the INREC, SORT and OUTREC control statements shown above, the output records are:

```
   CARRIE
DAVID
          FRANK
    MARTIN
      SAM
       VICKY
```

If we wanted the output to contain the sorted left-justified fields, we could use these control statements:

```
  INREC BUILD=(1,15,JFY=(SHIFT=LEFT))
  SORT FIELDS=(1,15,CH,A)
```

The output records would then be:

```
CARRIE
DAVID
FRANK
MARTIN
SAM
VICKY
```

Example 2

```
  OPTION COPY
  OUTREC OVERLAY=(11:11,16,JFY=(SHIFT=RIGHT,LEAD=C'(',
    TRAIL=C')',LENGTH=18))
```

This example illustrates how you can right-justify fields within your records.

The 50-byte FB input records might look like this:

```
0001      9-1-632-731
0002      011-276-321-7836
0003      753-218-307
0004      528-314
```

Note that the second field has left-justified numeric values in various forms. We want to right-justify these values and surround them with a left paren and right paren.

The 50-byte FB output records look like this:

```
0001           (9-1-632-731)
0002      (011-276-321-7836)
0003           (753-218-307)
0004               (528-314)
```

We use OUTFIL OVERLAY to limit the changes to the second field. We use JFY to right-justify the second field with surrounding parens. SHIFT=RIGHT shifts the characters to the right. LEAD=C'(' adds a left paren before the first non-blank character. TRAIL=C')' adds a right paren after the last non-blank character. LENGTH=18 increases the output length by 2 bytes to allow for the parens (overriding the default of 16 from the input length).

Left-Squeeze and Right-Squeeze (SQZ)

Introduction

The BUILD, OVERLAY, IFTHEN BUILD and IFTHEN OVERLAY operands of the INREC, OUTREC and OUTFIL statements now allow you to use a new SQZ option to left-squeeze or right-squeeze the data in a field. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-squeezed field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally for SQZ, specific characters can be changed to blanks before squeezing begins, a leading string can be inserted, a trailing string can be inserted, a string can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank, blanks can be kept as is between paired apostrophes or paired quotes, and the output length can be changed.

Tutorial

Suppose you had input records that looked like this

```
<tag>   History  </tag>
   <tag> Psychology      </tag>
 <tag>      Business </tag>
<tag>Biology</tag>
  <tag>   Science  </tag>
```

If you want to remove the white space (blanks), you can use the following statements to left-squeeze the data:

```
  OPTION COPY
  OUTREC FIELDS=(1,40,SQZ=(SHIFT=LEFT))
```

1,40,SQZ=(SHIFT=LEFT) indicates that you want to left-squeeze the data in positions 1-40 by removing all of the blanks, shifting the remaining characters to the left and padding on the right with blanks if needed. The results produced for this OUTREC statement are:

```
<tag>History</tag>
<tag>Psychology</tag>
<tag>Business</tag>
<tag>Biology</tag>
<tag>Science</tag>
```

Alternatively, you can use the following statements to right-squeeze the data:

```
  OPTION COPY
  OUTREC FIELDS=(1,40,SQZ=(SHIFT=RIGHT))
```

1,40,SQZ=(SHIFT=RIGHT) indicates that you want to right-squeeze the data in positions 1-40 by removing all of the blanks, shifting the remaining characters to the right and padding on the left with blanks if needed. The results produced for this OUTREC statement are:

```
                    <tag>History</tag>
                <tag>Psychology</tag>
                  <tag>Business</tag>
                   <tag>Biology</tag>
                   <tag>Science</tag>
```

DFSORT's squeeze feature also lets you add a leading string, a trailing string, or both, to the data. For example, you can use the following statements to surround the left-squeezed data with '<tag1>' and '</tag1>':

```
  OPTION COPY
  OUTREC FIELDS=(1,40,SQZ=(SHIFT=LEFT,LEAD=C'<tag1>',
     TRAIL=C'</tag1>'))
```

The results produced for this OUTREC statement are:

```
<tag1><tag>History</tag></tag1>
<tag1><tag>Psychology</tag></tag1>
<tag1><tag>Business</tag></tag1>
<tag1><tag>Biology</tag></tag1>
<tag1><tag>Science</tag></tag1>
```

LEAD=string specifies the leading string as a character or hexadecimal constant (1 to 50 bytes). TRAIL=string specifies the trailing string as a character or hexadecimal constant (1 to 50 bytes).

Notice that when you left-squeeze, the trailing string is placed directly to the right of the last non-blank character in your data.

If you want to right-squeeze instead of left-squeeze, you can use the following statements:

```
  OPTION COPY
  OUTREC FIELDS=(1,40,SQZ=(SHIFT=RIGHT,LEAD=C'<tag1>',
     TRAIL=C'</tag1>'))
```

The results produced for this OUTREC statement are:

```
         <tag1><tag>History</tag></tag1>
      <tag1><tag>Psychology</tag></tag1>
        <tag1><tag>Business</tag></tag1>
         <tag1><tag>Biology</tag></tag1>
         <tag1><tag>Science</tag></tag1>
```

Notice that when you right-squeeze, the leading string is placed directly to the left of the first non-blank character in your data.

If your leading or trailing string causes the output field to be longer than the input field, you will lose characters. In order to avoid that, you can increase the length of the output field with the LENGTH parameter as discussed previously under "Left-Justifying and Right-Justifying Data".

The squeeze feature can also be used to remove characters other than blanks from your data. Suppose you had input records that looked like this:

```
<tag>   (History) </tag>
   <tag> (Psychology)    </tag>
 <tag>      (Business) </tag>
<tag>(Biology)</tag>
  <tag>   (Science)  </tag>
```

You can use the following statements to remove the '(' and ')' characters before you left-squeeze the data:

```
  OPTION COPY
  OUTREC FIELDS=(1,40,SQZ=(SHIFT=LEFT,PREBLANK=C'()'))
```

PREBLANK=list specifies a list of characters you want to replace with blanks before DFSORT starts to squeeze the data. You can specify the list as a character or hexadecimal constant (1 to 10 bytes). Remember that each character in the list is independent of the other characters. For example, PREBLANK=C'*/' replaces each '*' character and '/' character with a blank before squeeze procssing begins (for example, character sequences of /*, //*, */, // and * are all replaced with blanks). The results produced for this OUTREC statement are:

```
<tag>History</tag>
<tag>Psychology</tag>
<tag>Business</tag>
<tag>Biology</tag>
<tag>Science</tag>
```

You could use the following statements to right-squeeze the data and remove the '(' character and ')' character:

```
OPTION COPY
OUTREC FIELDS=(1,40,SQZ=(SHIFT=RIGHT,PREBLANK=C'()'))
```

The results produced for this OUTREC statement are:

```
                    <tag>History</tag>
                <tag>Psychology</tag>
                  <tag>Business</tag>
                   <tag>Biology</tag>
                   <tag>Science</tag>
```

The squeeze feature can be used to replace groups of blanks between the first nonblank and last nonblank with other characters. Suppose you had input records that looked like this:

```
Manufacturing    California    +100000
Research         Arizona        +50000
Marketing        Texas          +75000
```

You could use the following statements to create comma separated variable records:

```
OPTION COPY
OUTREC FIELDS=(1,80,SQZ=(SHIFT=LEFT,MID=C','))
```

The results produced for this OUTREC statement are:

```
Manufacturing,California,+100000
Research,Arizona,+50000
Marketing,Texas,+75000
```

MID=string specifies the string to replace removed blanks or PREBLANK characters as a character or hexadecimal constant (1 to 10 bytes).

If you want DFSORT to "ignore" blanks and PREBLANK characters between pairs of quotes, you can use PAIR=QUOTE with SQZ. Suppose you had input records that looked like this:

```
"Computer Science A+"       +123
"Ancient Civilization B-"   +521
"Sanskrit A-"               -263
```

You could use the following statements to "protect" the blanks, + and - signs inside the paired quotes while removing them outside the paired quotes, and leave only one blank between the two squeezed fields:

```
OPTION COPY
OUTREC FIELDS=(1,80,SQZ=(SHIFT=LEFT,PAIR=QUOTE,
  PREBLANK=C'+-',MID=C' '))
```

The results produced for this OUTREC statement are:

```
"Computer Science A+" 123
"Ancient Civilization B-" 521
"Sanskrit A-" 263
```

If you want DFSORT to "ignore" blanks and PREBLANK characters between pairs of apostrophes, you can use PAIR=APOST with SQZ. Suppose you had input records that looked like this:

```
'Computer Science A+'       +123
'Ancient Civilization B-'   +521
'Sanskrit A-'               -263
```

You could use the following statements to "protect" the blanks, + and - signs inside the paired apostrophes while removing them outside the paired apostrophes, and leave only one blank between the two squeezed fields:

```
OPTION COPY
OUTREC FIELDS=(1,80,SQZ=(SHIFT=LEFT,PAIR=APOST,
  PREBLANK=C'+-',MID=C' '))
```

The results produced for this OUTREC statement are:

```
'Computer Science A+' 123
'Ancient Civilization B-' 521
'Sanskrit A-' 263
```

Detailed Description and Syntax

You can use p,m,squeeze in the BUILD and OVERLAY operands of the INREC, OUTREC and OUTFIL statements.

p,m,squeeze

specifies that a left-squeezed or right-squeezed input field is to appear in the reformatted record. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-squeezed field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

- specific characters can be changed to blanks before squeezing begins
- a leading string can be inserted
- a trailing string can be inserted
- a string (for example, a comma delimiter) can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank
- blanks can be kept as is between paired apostrophes ('AB CD EF') or paired quotes ("AB CD EF")
- the output length can be changed (it's equal to the input length by default)

p,m specifies the starting position and length of the 1 to 32752 byte field to be squeezed. The field must not extend beyond position 32752.

squeeze specifies how the input field is to be squeezed for output.

SQZ

```
>>-SQZ=(-+-SHIFT=LEFT--+-+-----------+-+----------------+-------->
         '-SHIFT=RIGHT-' '-,LENGTH=n-' '-,PREBLANK=list-'

>--+--------------+-+-----------+-+--------------+--------------->
   '-,LEAD=string-' '-MID=string' '-TRAIL=string-'

>--+-------------+---------------------------)----------------->
   +-,PAIR=APOST-+
   '-,PAIR=QUOTE-'
```

Note: For clarity, in the examples below, b is used to represent a blank in the input fields.

SHIFT=LEFT

specifies that a left-squeezed input field is to appear in the reformatted record. All blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For example, with:

```
1,15,SQZ=(SHIFT=LEFT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
ABCDEFbbbbbbbbbb
```

m is used for the output field length unless LENGTH=n is specified. If the left-squeezed input field is shorter than the output field length, blanks are inserted on the right. If the left-squeezed input field is longer than the output field length, characters are truncated on the right.

SHIFT=RIGHT

specifies that a right-squeezed input field is to appear in the reformatted record. All blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. For example, with:

```
1,15,SQZ=(SHIFT=RIGHT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
bbbbbbbbbABCDEF
```

m is used for the output field length unless LENGTH=n is specified. If the right-squeezed input field is shorter than the output field length, blanks are inserted on the left. If the right-squeezed input field is longer than the output field length, characters are truncated on the left.

LENGTH-n

specifies the length of the squeezed output field. The value for n must be between 1 and 32752. If LENGTH=n is not specified, m (the input field length) is used for the length of the squeezed output field.

PREBLANK=list

specifies a list of one or more characters to be changed to blanks **before** squeeze processing begins. Each nonblank character in the input field that matches a character in the list is replaced by a blank character.

list can be 1 to 10 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

Each character in the list is treated as one character to be preblanked. Thus PREBLANK=C'*' specifies an asterisk is to be preblanked, and PREBLANK=C',;$' specifies a comma, a semicolon and a dollar sign are each to be preblanked. For example, let's say we have an input field of:

```
**b<*ABbC<>D*bEF>*b**
```

If we specify:

```
1,21,SQZ=(SHIFT=LEFT,PREBLANK=C'*',LENGTH=12)
```

each asterisk is changed to a blank before left-squeeze processing begins. The output field is:

```
<ABC<>DEF>bb
```

If we specify:

```
1,21,SQZ=(SHIFT=RIGHT,PREBLANK=C'*<>')
```

each asterisk, less than sign and greater than sign is changed to a blank before right-squeeze processing begins. The output field is:

```
bbbbbbbbbbbbbbbABCDEF
```

LEAD=string

Specifies a string to be inserted in the output field before the first nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,SQZ=(SHIFT=RIGHT,LEAD=C'XYZ')
```

the output field is:

```
bbbbXYZABCE
```

When we add characters with LEAD=string, it's often necessary to specify LENGTH=n to avoid truncation. For additional information on this, see "LEAD=string" for JFY above.

MID=string

Specifies a string to be inserted in the output field wherever one or more blanks is removed between the first nonblank and the last nonblank.

string can be 1 to 10 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABbbCbbbDbE
```

If we specify:

```
1,12,SQZ=(SHIFT=LEFT,MID=C',')
```

the output field is:

```
AB,C,D,Ebbbb
```

When we add characters with MID=string, it's often necessary to specify LENGTH=n to avoid truncation. For example, let's say we have an input field of:

```
AbBbC
```

If we specify:

```
1,5,SQZ=(SHIFT=LEFT,MID=C'XY')
```

the output field is:

```
AXYBX
```

Since the output field length is defaulted to the input field length of 5, the resulting 7 characters (AXYBXYC) are truncated on the right to 5 characters (AXYBX) for output. If we instead specify:

```
1,5,SQZ=(SHIFT=LEFT,MID=C'XY',LENGTH=7)
```

the output field is:

```
AXYBXYC
```

LENGTH=7 increases the output field to acommodate the MID strings and truncation is prevented.

TRAIL=string

Specifies a string to be inserted in the output field after the last nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,SQZ=(SHIFT=LEFT,LEAD=X'7D',TRAIL=X'7D')
```

the output field is:

```
'ABCE'bbbbb
```

When we add characters with TRAIL=string, it's often necessary to specify LENGTH=n to avoid truncation. For additional information on this, see "TRAIL=string" for JFY above.

PAIR=APOST

Specifies that blanks and PREBLANK characters between apostrophe (') pairs are to be kept as is. Use PAIR=APOST when you have literals that should not be squeezed. For example, let's say we have an input field of:

```
b*b'ABbb*''*C'bbb'D*Ebb'*
```

If we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',')
```

all of the blanks and asterisks, including those inside the apostrophe pairs, are squeezed out. The output field is:

```
'AB,'',C','D,E,'bbbbbbbbb
```

However, if we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',',
          PAIR=APOST)
```

only the blanks and asterisks outside of the apostrophe pairs are squeezed out. The output field is:

```
'ABbb*''*C','D*Ebb'bbbbbb
```

If an apostrophe is specified in the PREBLANK list (for example, PREBLANK=C"' or PREBLANK=X'7D'), it is ignored, that is, an apostrophe in the input field is not replaced by a blank. So if you want each apostrophe to be replaced by a blank, do not specify PAIR=APOST.

For SHIFT=LEFT, scanning is left to right. If an apostrophe is found with no subsequent apostrophe, all of the characters from the apostrophe to the end of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY'ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=LEFT,PAIR=APOST)
```

there's an unpaired apostrophe, so all of the characters from the apostrophe to the end of the input field are ignored. The output field is:

```
XY'ABCbbbDbbbbb
```

For SHIFT=RIGHT, scanning is right to left. If an apostrophe is found with no previous apostrophe, all of the characters from the apostrophe to the beginning of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY'ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=RIGHT,PAIR=APOST)
```

there's an unpaired apostrophe, so all of the characters from the apostrophe to the beginning of the input field are ignored. The output field is:

```
bbbbbbbXbY'ABCD
```

PAIR=QUOTE

Specifies that blanks and PREBLANK characters between quote (") pairs are to be kept as is. Use PAIR=QUOTE when you have literals that should not be squeezed. For example, let's say we have an input field of:

```
b*b"ABbb*""*C"bbb"D*Ebb"*
```

If we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',')
```

all of the blanks and asterisks, including those inside the quote pairs, are squeezed out. The output field is:

```
"AB,"",C","D,E,"bbbbbbbbb
```

However, if we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',',
          PAIR=QUOTE)
```

only the blanks and asterisks outside of the quote pairs are squeezed out. The output field is:

```
"ABbb*""*C","D*Ebb'bbbbbb
```

If a quote is specified in the PREBLANK list (for example, PREBLANK=C"' or PREBLANK=X'7F'), it is ignored, that is, a quote in the input field is not replaced by a blank. So if you want each quote to be replaced by a blank, do not specify PAIR=QUOTE.

For SHIFT=LEFT, scanning is left to right. If a quote is found with no subsequent quote, all of the characters from the quote to the end of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY"ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=LEFT,PAIR=QUOTE)
```

there's an unpaired quote, so all of the characters from the quote to the end of the input field are ignored. The output field is:

```
XY"ABCbbbDbbbbb
```

For SHIFT=RIGHT, scanning is right to left. If a quote is found with no previous quote, all of the characters from the quote the beginning of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY"ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=RIGHT,PAIR=QUOTE)
```

there's an unpaired quote, so all of the characters from the quote to the beginning of the input field are ignored. The output field is:

```
bbbbbbbXbY"ABCD
```

Sample Syntax:

```
INREC BUILD=(5:16,20,SQZ=(SHIFT=LEFT,PAIR=QUOTE,PREBLANK=C'<>'))
```

Example 1

```
OPTION COPY
OUTFIL BUILD=(1,80,SQZ=(SHIFT=LEFT,PAIR=APOST,MID=C','))
```

This example illustrates how you can use SQZ to create FB output records with comma separated values from FB input records containing fields in fixed positions.

The 80-byte FB input records might look like this:

```
'John Lewis'         -12.83  'Research'
'Ted Blank'         +128.37  'Manufacturing'
'Marilyn Carlson'   -282.83  'Technical Support'
'Rex Otis'            +2.83  'Marketing'
```

Note that the data has three fields in fixed positions. The first field is a character string surrounded by apostrophes. The second field is a numeric value. The third field is another character string surrounded by apostrophes.

The 80-byte FB output records are in the form of comma separated values as follows:

```
'John Lewis',-12.83,'Research'
'Ted Blank',+128.37,'Manufacturing'
'Marilyn Carlson',-282.83,'Technical Support'
'Rex Otis',+2.83,'Marketing'
```

We use OUTFIL BUILD to build the output records with comma separated values. We use SQZ to squeeze out the blanks between the fields, shift the remaining characters to the left and insert commas between the fields. SHIFT=LEFT shifts the characters to the left. PAIR=APOST ensures that blanks within paired apostrophes are not squeezed out (for example, we want to keep the blank in the 'John Lewis' field.) MID=C',' inserts a comma for each group of blanks removed between the fields (for example, between 'John Lewis' and -12.83).

Example 2

```
  OPTION COPY
  OUTFIL IFTHEN=(WHEN=INIT,
    OVERLAY=(5:5,18,JFY=(SHIFT=LEFT,LEAD=C'''',TRAIL=C''''),
             34:34,19,JFY=(SHIFT=LEFT,LEAD=C'''',TRAIL=C''''))),
   IFTHEN=(WHEN=INIT,
    BUILD=(1,4,5,60,SQZ=(SHIFT=LEFT,PAIR=APOST,MID=C','))),
   VLTRIM=C' '
```

This example illustrates how you can use JFY and SQZ to create VB output records with comma separated values from VB input records containing fields in fixed positions.

The 84-byte VB input records might look like this (4-byte RDW followed by data):

```
Length   |  Data
    45       John Lewis          -12.83  Research
    50       Ted Blank          +128.37  Manufacturing
    54       Marilyn Carlson    -282.83  Technical Support
    51       Rex Otis             +2.83  Marketing
```

Note that the data has three fields in fixed positions. The first field is a character string. The second field is a numeric value. The third field is another character string. (In the previous example, the character strings were surrounded by apostrophes; in this example the apostrophes must be added around the character strings.)

The 84-byte VB output records are in the form of comma separated values as follows:

```
Length   |  Data
    34     'John Lewis',-12.83,'Research'
    39     'Ted Blank',+128.37,'Manufacturing'
    49     'Marilyn Carlson',-282.83,'Technical Support'
    32     'Rex Otis',+2.83,'Marketing'
```

We use OUTFIL IFTHEN to ensure that short records are padded on the right with blanks. (OUTFIL BUILD would terminate due to the short records.) WHEN=INIT reformats every record. OVERLAY surrounds the character strings with apostrophes. BUILD builds the output records with comma separated values.

We use JFY to surround the character strings with apostrophes without removing embedded blanks (for example, John Lewis is changed to 'John Lewis'). After the first IFTHEN, the records look like this:

```
Length   |  Data
    52     'John Lewis'          -12.83 'Research'
    52     'Ted Blank'          +128.37 'Manufacturing'
    54     'Marilyn Carlson'    -282.83 'Technical Support'
    52     'Rex Otis'             +2.83 'Marketing'
```

We then use SQZ to squeeze out the blanks between the fields, shift the remaining characters to the left and insert commas between the fields. SHIFT=LEFT shifts the characters to the left. PAIR=APOST ensures that blanks within paired apostrophes are not squeezed out (for example, we want to keep the blank in the 'John Lewis' field.) MID=C',' inserts a comma for each group of blanks removed between the fields (for example, between 'John Lewis' and -12.83).

Finally, we use VLRTIM=C' ' to remove trailing blanks at the end of each VB record by adjusting its length appropriately.

Example 3

```
  OPTION COPY
  OUTFIL REMOVECC,
   HEADER1=(C'<?xml version="1.0"?>',/,
     3:C'<booklist>'),
   BUILD=(5:C'<book>',/,
     7:1,20,JFY=(SHIFT=LEFT,LEAD=C'<title>',TRAIL=C'</title>',
       LENGTH=36),/,
     7:24,15,SQZ=(SHIFT=LEFT,LEAD=C'<author>',MID=C', ',
       TRAIL=C'</author>',LENGTH=33),/,
     5:C'</book>'),
   TRAILER1=(3:C'</booklist>')
```

This example illustrates how you can use JFY and SQZ to generate XML statements from FB input records.

The 40-byte FB input records might look like this:

```
Modern Poetry          Friedman    KR
Intro to Computers     Chatterjee  CL
Marketing              Maxwell     G
```

Note that the data has three character fields.

The 42-byte FB output records look like this:

```
<?xml version="1.0"?>
  <booklist>
    <book>
      <title>Modern Poetry</title>
      <author>Friedman, KR</author>
    </book>
    <book>
      <title>Intro to Computers</title>
      <author>Chatterjee, CL</author>
    </book>
    <book>
      <title>Marketing</title>
      <author>Maxwell, G</author>
    </book>
  </booklist>
```

We use OUTFIL HEADER1 to generate the xml and booklist starting tags that precede the set of tags for each record. We use OUTFIL BUILD to generate the set of tags for each record as follows:

- A constant is used to generate the book starting tag.
- JFY is used to generate the title tags and data from the first input field. LEAD generates the title starting tag before the input field from the record. TRAIL generates the title ending tag after the last nonblank character from the input field. JFY keeps embedded blanks between the first nonblank character and the last nonblank character of the input field. LENGTH ensures that the addition of the LEAD and TRAIL strings does not cause truncation by increasing the output length to 36 bytes (overriding the default of 20 bytes from the input field).
- SQZ is used to generate the author tags and data from the second and third input fields. LEAD generates the author starting tag before the input field from the record. MID replaces the blanks between the second and third input fields with a comma and one blank. TRAIL generates the author ending tag afer the last nonblank character from the input fields. LENGTH ensures that the addition of the LEAD, MID and TRAIL strings does not cause truncation by increasing the output length to 33 bytes (overriding the default of 15 bytes from the input field).

We use OUTFIL TRAILER1 to generate the booklist ending tag that follows the set of tags for each record.

Parsing Variable Fields (PARSE, %nn)

Introduction

PARSE and IFTHEN PARSE are new INREC, OUTREC and OUTFIL operands that allow you to extract variable position/length fields into fixed-length parsed fields defined as %nn fields. PARSE gives you powerful new capabilities for handling variable fields such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and many other types.

You can use various PARSE options to define the rules for extracting variable fields into up to one hundred %nn fixed-length parsed fields (%00-%99), and then use these %nn fields where you can use p,m (fixed) fields in BUILD, OVERLAY, IFTHEN BUILD and IFTHEN OVERLAY. You can edit, convert, justify, squeeze, translate, and do arithmetic with %nn fields.

You can even set up DFSORT symbols for your %nn fields as discussed in "Symbols for %nn Parsed Fields (sym,%nn)" below.

Tutorial

Chapter 5 "Reformatting Records" of *z/OS DFSORT: Getting Started* describes how you can reformat records with **fixed fields** (p,m), that is, fields that start in the same position and have the same length in every record. This tutorial describes how you can now reformat records in similar ways with **variable fields**, that is, fields that have different starting positions and lengths in different records, such as comma separated values (CSV).

Using %nn Parsed Fields with BUILD and OVERLAY

There are many types of variable position/length fields such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and so on. For example, you might have four records with comma separated values as follows:

```
Wayne,M,-53,-1732,Gotham
Summers,F,+7258,-273,Sunnydale
Kent,M,+213,-158,Metropolis
Prince,F,-164,+1289,Gateway
```

Note that each record has five variable fields separated by commas. The fields do not start and end in the same position in every record and have different lengths in different records, so you could not just specify the starting position and length (p,m) for any of these fields in a BUILD or OVERLAY operand of the INREC, OUTREC or OUTFIL statement. But you can use the PARSE operand of an INREC, OUTREC or OUTFIL statement to define rules that tell DFSORT how to extract the relevant data from each variable input field into a fixed parsed field, and then use the fixed parsed fields in a BUILD or OVERLAY operand as you would use fixed input fields.

You define a parsed field for converting a variable field to a fixed parsed field using a %nn name where nn can be 00 to 99. You can define and use up to 100 parsed fields per run. Each %nn parsed field must be defined only once. A %nn parsed field must be defined in a PARSE operand **before** it is used in a BUILD or OVERLAY operand.

Suppose you wanted to reformat the CSV records to produce these output records:

```
Wayne        -178.5     Gotham
Summers       698.5     Sunnydale
Kent            5.5     Metropolis
Prince        112.5     Gateway
```

You can use the following OUTREC statement to parse and reformat the variable fields:

```
 OUTREC PARSE=(%01=(ENDBEFR=C',',FIXLEN=8),
               %=(ENDBEFR=C','),
               %03=(ENDBEFR=C',',FIXLEN=5),
               %04=(ENDBEFR=C',',FIXLEN=5),
               %05=(FIXLEN=10)),
    BUILD=(%01,14:%03,SFF,ADD,%04,SFF,EDIT=(SIIT.T),SIGNS=(,-),
           25:%05)
```

The **PARSE** operand defines how each variable field is to be extracted to a fixed parsed field as follows:

- The **%01** parsed field is used to extract the first variable field into an 8-byte fixed parsed field. ENDBEFR=C',' tells DFSORT to stop extracting data at the byte before the next comma (the comma after the first variable field). FIXLEN=8 tells DFSORT that the %01 parsed field is 8 bytes long. Thus, for the first record, DFSORT extracts Wayne into the 8-byte %01 parsed field. Since Wayne is only 5 characters, but the %01 parsed field is 8 bytes long, DFSORT pads the %01 parsed field on the right with 3 blanks. ENDBEFR=C',' also tells DFSORT to skip over the comma after the first variable field before it parses the second variable field.
- The **%** parsed field is used to skip the second variable field without extracting anything for it. Since we don't want this field in the output record, we can use % to ignore it. Thus, for the first record, we ignore M. ENDBEFR=C',' tells DFSORT to skip over the comma after the second variable field before it parses the third variable field.
- The **%03** parsed field is used to extract the third variable field into a 5-byte fixed parsed field. ENDBEFR=C',' tells DFSORT to stop extracting data before the next comma (the comma after the third variable field). FIXLEN=5 tells DFSORT that the %03 parsed field is 5 bytes long. Thus, for the first record, DFSORT extracts -53 into the 5-byte %03 parsed field. Since -53 is only 3 characters, but the %03 parsed field is 5 bytes long, DFSORT pads the %03 parsed field on the right with 2 blanks. ENDBEFR=C',' also tells DFSORT to skip over the comma after the third variable field before it parses the fourth variable field.
- The **%04** parsed field is used to extract the fourth variable field into a 5-byte fixed parsed field. ENDBEFR=C',' tells DFSORT to stop extracting data before the next comma (the comma after the fourth variable field). FIXLEN=5 tells DFSORT that the %04 parsed field is 5 bytes long. Thus, for the first record, DFSORT extracts -1732 into the 5-byte %04 parsed field. Since -1732 is 5 characters, it fills up the the 5-byte %05 parsed field and padding is not needed. ENDBEFR=C',' also tells DFSORT to skip over the comma after the fourth variable field before it parses the fifth variable field.
- The **%05** parsed field is used to extract the fifth variable field into a 10-byte fixed parsed field. FIXLEN=10 tells DFSORT that the %05 parsed field is 10 bytes long. Thus, for the first record, DFSORT extracts Gotham and 4 blanks into the 10-byte %01 parsed field.

The **BUILD** operand uses the previously extracted fixed parsed fields to build the output record as follows:

- **%01** copies the 8-byte fixed-length data extracted from the first variable field to positions 1-8 of the output record. For the first record, positions 1-8 contain 'Wayne '.
- **14:%03,SFF,ADD,%04,SFF,EDIT=(SIIT.T),SIGNS=(,-)** adds the 5-byte fixed-length data extracted from the third variable field to the 5-byte fixed-length data extracted from the fourth variable field and places the 6-byte edited result in positions 14-19 of the output record. For the first record, positions 14-19 contain -178.5 (-53 + -1732 = -1785 edited to -178.5). Note that since the %03 and %04 parsed fields may be padded on the right with blanks, we must use the SFF format to handle the sign and digits correctly.

- **25:%05** copies the 10-byte fixed-length data extracted from the fifth variable field to positions 25-34 of the output record. For the first record, positions 25-34 contain 'Gotham ''.

Using %nn Parsed Fields with IFTHEN

If you have different variable position/length fields in different types of records, you can convert them to %nn fixed parsed fields and use them in IFTHEN clauses. If you define a %nn parsed field in a WHEN=INIT clause, you can use it in the IFTHEN BUILD or IFTHEN OVERLAY suboperand of that clause as well as in any IFTHEN clause that follows. If you define a %nn parsed field in a WHEN=(logexp), WHEN=ANY or WHEN=NONE clause, you can use it in the IFTHEN BUILD or IFTHEN OVERLAY suboperand of that clause.

Suppose you have the following input records:

```
1/Sam/Charlie;27
2/Bill/48
1/Frank/Vicky;02
1/William/Dale;86
2/Helen/15
```

Note that the records that start with '1' have one fixed field ('1') and three variable fields, whereas the records that start with '2' have one fixed field ('2') and two variable fields. The variable fields in each type of record do not start and end in the same position in every record and have different lengths in different records.

Suppose you wanted to reformat these input records to produce these output records:

```
1  Sam           Charlie     2.7
2  Bill                      4.8
1  Frank         Vicky       0.2
1  William       Dale        8.6
2  Helen                     1.5
```

You can use the following INREC statement to parse and reformat the fixed and variable fields:

```
  INREC IFOUTLEN=50,
        IFTHEN=(WHEN=INIT,
          PARSE=(%00=(ABSPOS=3,ENDBEFR=C'/',FIXLEN=8))),
        IFTHEN=(WHEN=(1,1,CH,EQ,C'1'),
          PARSE=(%01=(ABSPOS=3,STARTAFT=C'/',ENDBEFR=C';',
            FIXLEN=8),%02=(FIXLEN=2)),
          BUILD=(1,1,4:%00,18:%01,30:%02,ZD,EDIT=(T.T))),
        IFTHEN=(WHEN=(1,1,CH,EQ,C'2'),
          PARSE=(%03=(ABSPOS=3,STARTAFT=C'/',FIXLEN=2)),
          BUILD=(1,1,4:%00,30:%03,ZD,EDIT=(T.T)))
```

The PARSE suboperand of the WHEN=INIT clause uses the %00 parsed field to extract the first variable field into an 8-byte fixed parsed field for every record. ABSPOS=3 tells DFSORT to start at position 3. ENDBEFR=C'/' tells DFSORT to stop extracting data before the next slash (the slash after the first variable field). FIXLEN=8 tells DFSORT that the %00 parsed field is 8 bytes long. Thus, for the first record, DFSORT extracts Sam into the 8-byte %00 parsed field. Since Sam is only 3 characters, but the %00 parsed field is 8 bytes long, DFSORT pads the %00 parsed field on the right with 5 blanks. The WHEN=INIT clause defines %00 so it can be used for the clauses that follow. (%00 could be used in a BUILD or OVERLAY suboperand for this WHEN=INIT clause, but in this case, it just defines %00 for the other clauses.)

The PARSE suboperand of the first WHEN=(logexp) clause uses the %01 parsed field to extract the second variable field into an 8-byte fixed parsed field, and the %02 parsed field to extract the third variable field into a 2-byte fixed parsed field, for the '1' records.

For %01, ABSPOS=3 tells DFSORT to start at position 3. STARTAFT=C'/' tells DFSORT to start extracting data after the next slash (the slash after the first variable field). ENDBEFR=C';' tells DFSORT to stop extracting data before the next semicolon (the semicolon after the second variable field). FIXLEN=8 tells DFSORT that the %01 parsed field is 8 bytes long. Thus, for the first record, DFSORT extracts Charlie into the 8-byte %01 parsed field. Since Charlie is only 7 characters, but the %01 parsed field is 8 bytes long, DFSORT pads the %01 parsed field on the right with one blank.

For %02, FIXLEN=2 tells DFSORT that the %02 parsed field is 2 bytes long. Thus, for the first record, DFSORT extracts 27 into the 2-byte %02 parsed field.

The BUILD suboperand of the first WHEN=(logexp) clause uses the %00 parsed field defined by the WHEN=INIT clause and the %01 and %02 parsed fields defined by the first WHEN=(logexp) clause to build each '1' output record.

The PARSE suboperand of the second WHEN=(logexp) clause uses the %03 parsed field to extract the second variable field into a 2-byte fixed parsed field for the '2' records.

For %03, ABSPOS=3 tells DFSORT to start at position 3. STARTAFT=C'/' tells DFSORT to start extracting data after the next slash (the slash after the first variable field). FIXLEN=2 tells DFSORT that the %03 parsed field is 2 bytes long. Thus, for the second record, DFSORT extracts 48 into the 2-byte %03 parsed field.

The BUILD suboperand of the second WHEN=(logexp) clause uses the %00 parsed field defined by the WHEN=INIT clause and the %03 parsed field defined by the second WHEN=(logexp) clause to build each '2' output record.

Where You Can Use %nn Fields in BUILD and OVERLAY

z/OS Application Programming Guide provides complete details of the items where you can use p,m (fixed) fields in the BUILD or OVERLAY operands of the INREC, OUTREC and OUTFIL statements. Here's a list of the items where you can use %nn parsed fields in the same way you use p,m fields.

- %nn
- %nn,HEX
- %nn,TRAN=LTOU
- %nn,TRAN=UTOL
- %nn,TRAN=ALTSEQ
- %nn,f,edit
- %nn,f,to
- %nn,f in arexp,edit
- %nn,f in arexp,to
- %nn,Y2x
- %nn,Y2x,edit
- %nn,Y2x,to
- %nn,Y2x(c)
- %nn,Y2xP
- %nn,lookup
- %nn as set field in lookup

- %nn as set field in NOMATCH
- RESTART=(%nn) in SEQNUM

In addition you can use %nn,justify and %nn,squeeze in the same way you can use p,m,justify and p,m,squeeze (described earlier in this paper).

PARSE Parameters

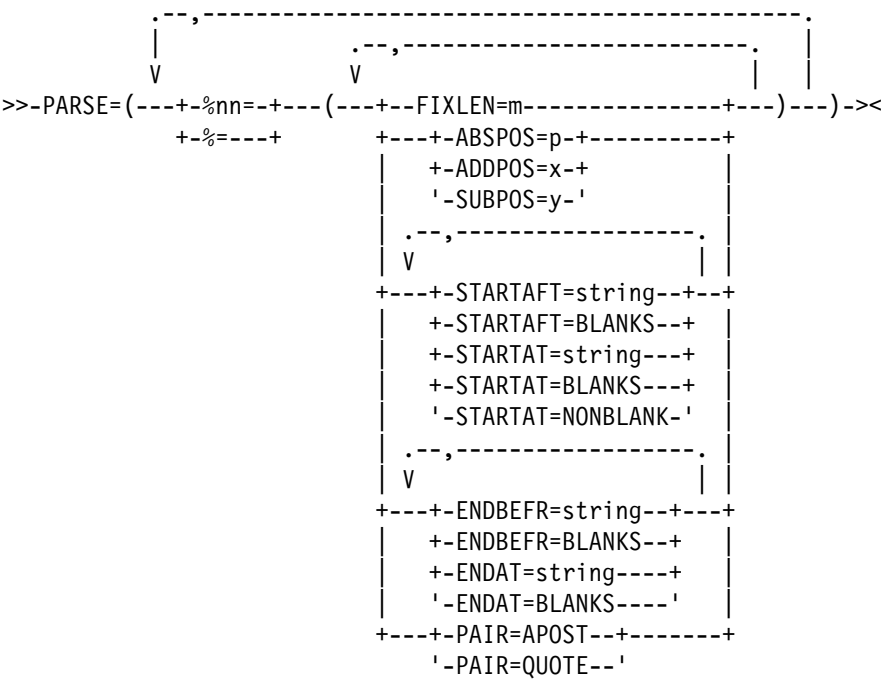
You can use the following parameters in PARSE to define the rules for extracting variable position/length data to %nn fixed parsed fields:

- **FIXLEN=m:** Specifies the length (m) of the fixed area to contain the extracted variable data for this %nn fixed parsed field.
- **ABSPOS=p:** Start extracting data at input position p.
- **ADDPOS=x:** Start extracting data at the current position + x.
- **SUBPOS=y:** Start extracting data at the current position - y.
- **STARTAFT=string:** Start extracting data at the byte after the end of the character or hexadecimal string.
- **STARTAFT=BLANKS:** Start extracting data after the end of the next group of blanks.
- **STARTAT=string:** Start extracting data at the first byte of the character or hexadecimal string.
- **STARTAT=BLANKS:** Start extracting data at the start of the first group of blanks.
- **STARTAT=NONBLANK:** Start extracting data at the next nonblank.
- **ENDBEFR=string:** Stop extracting data at the byte before the start of the character or hexadecimal string.
- **ENDBEFR=BLANKS:** Stop extracting data at the byte before the next group of blanks.
- **ENDAT=string:** Stop extracting data at the last byte of the character or hexadecimal string.
- **ENDAT=BLANKS:** Stop extracting data at the end of the next group of blanks.
- **PAIR=APOST:** Do not search for strings or blanks between apostrophe (') pairs.
- **PAIR=QUOTE:** Do not search for strings or blanks between quote (") pairs.

Detailed Description and Syntax

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

PARSE



This operand allows you to extract variable position/length fields into fixed parsed fields. Parsed fields (%nn) can be used where fixed position/length fields (p,m) can be used in the BUILD or OVERLAY operands.

PARSE can be used for many different types of variable fields including delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and so on. You can assign up to 100 %nn parsed fields (%00-%99) to the variable fields you want to extract.

Note that if all of the fields in your records have fixed positions and lengths, you don't need to use PARSE. But if any of the fields in your records have variable positions or lengths, you can use PARSE to treat them as fixed parsed fields in BUILD or OVERLAY. You can mix p,m fields (fixed fields) and %nn fields (parsed fields) in BUILD and OVERLAY.

For each %nn parsed field, you define where the data to be extracted starts and ends, and the length (m) of the fixed field to contain the extracted data. For example, if your input records contained CSV data as follows:

```
AA,BBBB,C,DDDDD
EEEEE,,F,GG
HHH,IIIII,JJ,K
```

and you wanted to reformat the data into fixed positions like this:

```
AA        BBBB      C         DDDDD
EEEEE               F         GG
HHH       IIIII     JJ        K
```

you could use this INREC statement:

```
  INREC PARSE=(%01=(ENDBEFR=C',',FIXLEN=5),
               %02=(ENDBEFR=C',',FIXLEN=5),
               %03=(ENDBEFR=C',',FIXLEN=5),
               %04=(FIXLEN=5)),
        BUILD=(1:%01,11:%02,21:%03,31:%04)
```

The PARSE operand:

- assigns %01 to the first parsed field, and defines it as starting at position 1 and ending before the next (first) comma with a fixed length of 5 bytes
- assigns %02 to the second parsed field, and defines it as starting after the (first) comma and ending before the next (second) comma with a fixed length of 5 bytes
- assigns %03 to the third parsed field, and defines it as starting after the (second) comma and ending before the next (third) comma with a fixed length of 5 bytes
- assigns %04 to the fourth parsed field, and defines it as starting after the (third) comma and ending after 5 bytes with a fixed length of 5 bytes

You can start extracting data at a specific position (ABSPOS=p), a relative position (ADDPOS=x or SUBPOS=y), after the start of one or more specific strings (STARTAFT=string) or blanks (STARTAFT=BLANKS), or at the start of one or more specific strings (STARTAT=string) or blanks (STARTAT=BLANKS) or a nonblank (STARTAT=NONBLANK). You can end extracting data before the start of one or more specific strings (ENDBEFR=string) or blanks (ENDBEFR=BLANKS), at the end of one or more specific strings (ENDAT=string) or blanks (ENDAT=BLANKS), or after a specifed number of bytes (FIXLEN=m without ENDBEFR or ENDAT).

For each record, each %nn and % parsed field you define is processed according to the suboperands you define, in the following order:

- ABSPOS or ADDPOS or SUBPOS
- STARTAFT, STARTAT and PAIR
- ENDBEFR, ENDAT and PAIR
- FIXLEN

As each %nn parsed field is processed, data is extracted according to the suboperands you specify.

In addition, as each %nn or % parsed field is processed, the Start Pointer for the next parsed field advances through the record according to the suboperands you specify. By default, for the first parsed field, the Start Pointer is set to position 1 for fixed-length records or to position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field uses the Start Pointer as set by the previous %nn parsed field. For any parsed field, the Start Pointer advances as follows depending on the suboperands specified, and may advance more than once (for example, if STARTAFT and ENDBEFR are both specified, or if STARTAT and FIXLEN are both specified):

- if ABSPOS=p is specified, the Start Pointer is set to position p.
- if ADDPOS=x is specified, the Start Pointer is incremented by x.
- if SUBPOS=y is specified, the Start Pointer is decremented by y.
- if STARTAFT=string or STARTAT=string is specified, the search for string starts at the Start Pointer. If string is found, the Start Pointer is set to to the byte after the end of the string. If string is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if STARTAFT=BLANKS or STARTAT=BLANKS is specified, the search for blanks starts at the Start Pointer. If a blank is found, the Start Pointer is set to the next nonblank. If a blank is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if STARTAT=NONBLANK is specified, the search for a nonblank starts at the Start Pointer. If a nonblank is found, the Start Pointer is set to the nonblank. If a nonblank is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if ENDBEFR=string or ENDAT=string is specified, the search for string starts at the Start Pointer. If string is found, the Start Pointer is set to to the byte after the end of the string. If string is not found, data for this parsed field is extracted up to the end of the record. Subsequent parsed fields contain all blanks.

- if ENDBEFR=BLANKS or ENDAT=BLANKS is specified, the search for blanks starts at the Start Pointer. If a blank is found, the Start Pointer is set to the next nonblank. If a blank is not found, data for this parsed field is extracted up to the end of the record. Subsequent parsed fields contain all blanks.
- If ENDBEFR and ENDAT are not specified and FIXLEN=m is specifed, the Start Pointer is set m bytes past the start of the extracted data.
- if the Start Pointer advances past the end of the record, PARSE processing stops.

For example, if you had these records as input:

```
11*+23  NEW  MAXCC=00   (Monica) 18
1*-6 OLD  MAXCC=04 (Joey)           -2735
232*+86342 SHR MAXCC=04  (Rachel)  0
```

and you wanted this output:

```
Monica    MAXCC=00  +000023  011     18
Joey      MAXCC=04  -000006  001  -2735
Rachel    MAXCC=04  +086342  232      0
```

you could use this OUTFIL statement:

```
  OUTFIL FNAMES=OUT1,
    PARSE=(%00=(ENDBEFR=C'*',FIXLEN=3),
      %01=(ENDBEFR=BLANKS,FIXLEN=6),
      %02=(STARTAT=C'MAX',FIXLEN=8),
      %03=(STARTAFT=C'(',ENDBEFR=C')',FIXLEN=6),
      %04=(STARTAFT=BLANKS,FIXLEN=5)),
    BUILD=(%03,11:%02,21:%01,SFF,M26,LENGTH=7,
           30:%00,UFF,M11,LENGTH=3,35:%04,JFY=(SHIFT=RIGHT))
```

For this example, the Start Pointer advances as follows for the first record:

- For %00: Set Start Pointer to position 1. Start searching for '*' end string in position 1. Extract '11' (per ENDBEFR=C'*'). Set Start Pointer after '*' (at '+').
- For %01: Start searching for end blank at the Start Pointer (at '+'). Extract '+23' (per ENDBEFR=BLANKS). Set Start Pointer after the blanks (at 'N').
- For %02: Start searching for 'MAX' start string at the Start Pointer (at 'N'). Extract 'MAXCC=00' (per FIXLEN=8). Increment Start Pointer by m (8) bytes from the start of the extracted string (at blank after '00').
- For %03: Start searching for '(' start string at Start Pointer (at blank after '00'). Set Start Pointer after '(' (at 'M'). Start searching for ')' end string at Start Pointer (at 'M'). Extract 'Monica' (per ENDBEFR=C')'). Set Start Pointer after ')' (at blank after ')').
- For %04: Start searching for start blank at the Start Pointer (at blank after ')'). Extract '18 ' (per FIXLEN=5).

If the Start Pointer advances past the end of the record, PARSE processing stops. For example, if you had these input records:

```
First=George Last=Washington
First=John Middle=Quincy Last=Adams
```

and you used this OUTREC statement:

```
  OUTREC PARSE=(%00=(STARTAFT=C'First=',ENDBEFR=C' ',FIXLEN=12),
           %01=(STARTAFT=C'Middle=',ENDBEFR=C' ',FIXLEN=12),
           %02=(STARTAFT=C'Last=',ENDBEFR=C' ',FIXLEN=12)),
    BUILD=(%00,%01,%02)
```

the %01 parsed field (Middle name) is not found in the first record, so PARSE processing stops and the %02 parsed field is set to blanks.

You can handle this kind of possible missing field situation by using IFTHEN PARSE instead of PARSE. For example:

```
OUTREC IFTHEN=(WHEN=INIT,
    PARSE=(%00=(STARTAFT=C'First=',ENDBEFR=C' ',FIXLEN=12))),
  IFTHEN=(WHEN=INIT,
    PARSE=(%01=(STARTAFT=C'Middle=',ENDBEFR=C' ',FIXLEN=12))),
  IFTHEN=(WHEN=INIT,
    PARSE=(%02=(STARTAFT=C'Last=',ENDBEFR=C' ',FIXLEN=12)),
  BUILD=(%00,%01,%02))
```

By default, the Start Pointer is set to 1 (F) or 5 (V) for each IFTHEN PARSE, so the missing middle name in the first record does not prevent the last name from being extracted.

You must define each %nn field with PARSE before you use it in BUILD or OVERLAY, so generally you would want to specify PARSE before BUILD or OVERLAY. If you define a %nn field with PARSE, but do not use it in BUILD or OVERLAY, data will not be extracted for that parsed field.

Each %nn parsed field (%00-%99) can only be defined once per run with PARSE, but can be used as many times as needed in BUILD or OVERLAY.

The %nn parsed fields defined for a particular source (INREC, OUTREC or OUTFIL) can only be used in the BUILD or OVERLAY operands for that source. Generally, you will only use PARSE for one source (for example, INREC) per run. But you could define a separate set of %nn parsed fields for each source, if needed. For example, you could define %21 and %22 for the INREC statement, %01, %02 and %03 for the OUTREC statement, %40 and %45 for OUTFIL statement 1, and %30, %31 and %32 for OUTFIL statement 2. You could then use %21 and %22 for INREC BUILD or OVERLAY, %02 and %03 for OUTREC BUILD or OVERLAY, %40 and %45 for OUTFIL statement 1 BUILD or OVERLAY, and %30, %31 and %32 for OUTFIL statement 2 BUILD or OVERLAY. Note that you could not, for example, use %21 and %22 for OUTREC BUILD or OVERLAY.

If you specify PARSE=(...),IFTHEN=(...) or IFTHEN=(...),PARSE=(...), DFSORT will terminate. If you need to specify PARSE with IFTHEN, specify it within one or more individual IFTHEN clauses, as appropriate. For details, see "PARSE in IFTHEN clauses" below.

%nn

Assigns %nn to a parsed field. The variable-length data defined for the parsed field is extracted to a fixed length area (padded with blanks or truncated as appropriate) and can be used where a p,m field can be used in BUILD or OVERLAY.

nn can be 00 to 99, but each %nn value can only be defined once per run. This gives you a maximum of 100 parsed fields per run.

%0 to %9 can be used, but will be treated as equivalent to %00 to %09. You can use %n and %0n interchangeably, but you cannot define both %n and %0n in the same run.

Sample Syntax

```
OUTFIL PARSE=(%99=(ABSPOS=6,STARTAT=C'KW=(',ENDAT=C')',FIXLEN=20),
  %03=(STARTAFT=BLANKS,FIXLEN=10)),
  OVERLAY=(1:1,5,%03,C'=',%99,1:1,36,SQZ=(SHIFT=LEFT))
```

%

Specifies a parsed field to be ignored. No data is extracted, but the starting point for the next parsed field advances according to the suboperands specified. Use % when you don't need the data from a particular field,

but you do need to get to the next field. For example, if we had the four CSV fields shown earlier as input, but we only wanted to extract the first and fourth fields, we could use this INREC statement:

```
INREC PARSE=(%01=(ENDBEFR=C',',FIXLEN=5),
             %=(ENDBEFR=C','),
             %=(ENDBEFR=C','),
             %04=(FIXLEN=5)),
      BUILD=(1:%01,11:%04,21:%01,HEX)
```

Data is extracted for %01 (first field) and %04 (fourth field), but not for % (second and third fields).

FIXLEN=m

Specifies the length (m) of the fixed area to contain the extracted variable-length data for this %nn fixed parsed field. m can be 1 to 32752. You must specify FIXLEN=m for each %nn parsed field. FIXLEN=m is optional for a % ignored field.

When %nn is specified in BUILD or OVERLAY, the variable-length data is extracted to the fixed area and then used for BUILD or OVERLAY processing. The fixed area always contains m bytes. If the extracted data is longer than m, the fixed area contains the first m bytes of the extracted data (the data is truncated on the right to m bytes). If the extracted data is shorter than m, the fixed area contains the extracted data padded with blanks on the right to m bytes. If no data is extracted, the fixed area contains all blanks.

The length m from FIXLEN=m for fixed parsed fields (%nn) corresponds to the length (m) for fixed fields (p,m). Keep in mind that %nn fields are left-aligned and may be padded on the right with blanks. You must ensure that the data in each %nn field and its length (m) are valid for the way you want to use that %nn field in BUILD or OVERLAY. For example, if you had these records as input:

```
123,8621
1,302
18345,17
```

and you wanted this output:

```
  1.23   86.21
  0.01    3.02
183.45    0.17
```

you could use this OUTREC statement:

```
OUTREC PARSE=(%00=(ENDBEFR=C',',FIXLEN=5),
              %01=(FIXLEN=5)),
       BUILD=(%00,UFF,EDIT=(IIT.TT),2X,%01,UFF,EDIT=(IIT.TT))
```

Note that the 5-byte extracted parsed fields for %00 are left-aligned and padded with blanks (b) like this:

```
123bb
1bbbb
18345
```

and the 5-byte extracted parsed fields for %01 are left-aligned and padded with blanks (b) like this:

```
8621b
302bb
17bbb
```

Thus, you cannot use numeric formats like ZD or FS for these parsed fields, but you can use UFF. Note also that you must ensure that m from FIXLEN=m for each %nn parsed field is valid for the specific BUILD or OVERLAY item in which you use %nn. For example, you could use 1-44 as m for UFF, but you couldn't use 45 as m.

If ENDBEFR and ENDAT are not specified, the Start Pointer is set m bytes past the start of the extracted data. For example, if you had this record as input:

```
MAX=(ABCDEF)
```

and you specified:

```
OUTFIL PARSE=(%00=(STARTAT=C'MAX',FIXLEN=8),
     %01=(FIXLEN=3),%02=(FIXLEN=1)),
   BUILD=(%00,12:%01,20:%02)
```

The output record would be:

```
MAX=(ABC   DEF     )
```

Sample Syntax

```
INREC PARSE=(%00=(STARTAFT=C'KW=',FIXLEN=12)),
    BUILD=(%00)
```

ABSPOS=p

Sets the Start Pointer for this parsed field to p. p can be 1 to 32752. By default, the Start Pointer for the first %nn parsed field is position 1 for fixed-length records or position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field is the Start Pointer set by the previous %nn field. You can use ABSPOS=p to set the Start Pointer to position p to override the default Start Pointer. If the resulting Start Pointer is less than position 5 for variable-length records, it will be set to position 5.

Example

If your input is:

```
****|BB|CCCC|
****|EEEE|FF|
```

and you wanted to reformat the data into fixed positions like this:

```
**** BB   CCCC
**** EEEE FF
```

you could use this OUTFIL statement:

```
OUTFIL PARSE=(%01=(ABSPOS=6,ENDBEFR=C'|',FIXLEN=4),
      %02=(ENDBEFR=C'|',FIXLEN=4)),
    BUILD=(1,4,X,%01,X,%02)
```

The initial Start Pointer for the %01 parsed field is set to position 6 instead of to position 1.

ADDPOS=x

Increments the Start Pointer for this parsed field by x. x can be 1 to 32752. By default, the Start Pointer for the first %nn parsed field is position 1 for fixed-length records or position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field is the Start Pointer set by the previous %nn field. You can use ADDPOS=x to increment the Start Pointer by x to override the default Start Pointer.

Sample Syntax

```
OUTREC PARSE=(%00=(ENDAT=C'||',FIXLEN=10),
     %01=(ADDPOS=5,STARTAFT=C';',FIXLEN=6)),
   BUILD=(%00,TRAN=ALTSEQ,%01)
```

SUBPOS=y

Decrements the Start Pointer for this parsed field by y. y can be 1 to 32752. By default, the Start Pointer for the first %nn parsed field is position 1 for fixed-length records or position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field is the Start Pointer set by the previous %nn field. You can use SUBPOS=y to decrement the Start Pointer by x to override the default Start Pointer. If the resulting Start

Pointer is less than position 1 for fixed-length records, it will be set to position 1. If the resulting Start Pointer is less than position 5 for variable-length records, it will be set to position 5.

Sample Syntax

```
OUTFIL PARSE=(%00=(ENDBEFR=C'|;|',FIXLEN=10),
     %01=(SUBPOS=1,STARTAT=C'|',ENDAT=C'|',FIXLEN=12)),
   BUILD=(%00,TRAN=ALTSEQ,%01)
```

STARTAFT=string

Data is to be extracted for this parsed field starting after the last byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is not extracted for this parsed field or any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field starting at the byte after the end of the string, and the Start Pointer is set to the byte after the end of the string.

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS Application Program Guide* for details of coding character and hexadecimal string constants.

Example

If your input is:

```
         MAX=25,832        MIN=2,831  $4
  MAX=1,275    MIN=17 %3
```

and you wanted your output to contain MAX-MIN, you could use this INREC statement:

```
INREC PARSE=(%00=(STARTAFT=C'MAX=',ENDBEFR=X'40',FIXLEN=6),
    %01=(STARTAFT=C'MIN=',ENDBEFR=X'40',FIXLEN=6)),
  OVERLAY=(50:C'DELTA:  ',%00,UFF,SUB,%01,UFF,TO=ZD,LENGTH=8)
```

STARTAFT=BLANKS

Data is to be extracted for this parsed field starting at the first nonblank after one or more blanks. The search for a blank begins at the Start Pointer. If a blank is not found, data is not extracted for this parsed field or any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field starting at the first nonblank, and the Start Pointer is set to the first nonblank. *Example*

If your input is:

```
        Frank     D28
  Vicky                   D52
```

and you wanted your output to be:

```
Frank        D28
Vicky        D52
```

you could use this OUTREC statement:

```
OUTREC PARSE=(%00=(STARTAFT=BLANKS,FIXLEN=8),
    %01=(STARTAFT=BLANKS,FIXLEN=3)),
  BUILD=(%00,14:%01)
```

STARTAT=string

Data is to be extracted for this parsed field starting at the first byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is not extracted for this parsed field or any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field starting at the first byte of the string, and the Start Pointer is set to the byte after the end of the string.

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS Application Program Guide* for details of coding character and hexadecimal string constants.

Example

If your input is:

```
         MAX=58321        MIN=00273
   MAX=01275    MIN=00017
```

and you wanted your output to be:

```
MAX=58321 MIN=00273
MAX=01275 MIN=00017
```

you could use this INREC statement:

```
  INREC PARSE=(%00=(STARTAT=C'MAX=',FIXLEN=9),
      %01=(STARTAT=C'MIN=',FIXLEN=9)),
    BUILD=(%00,X,%01)
```

STARTAT=BLANKS

Data is to be extracted for this parsed field starting at the first blank. The search for a blank begins at the Start Pointer. If a blank is not found, data is not extracted for this parsed field or any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field starting at the first blank, and the Start Pointer is set to the first nonblank.

STARTAT=NONBLANK

Data is to be extracted for this parsed field starting at the first nonblank. The search for a nonblank begins at the Start Pointer. If a nonblank is not found, data is not extracted for this parsed field or any subsequent parsed fields. If a nonblank is found, data is extracted to the fixed area for this parsed field starting at the nonblank, and the Start Pointer is set to the nonblank.

Example

If your input is:

```
Frank     D28
   Victoria           D52
  Holly D15
Roberta     D52
```

and you wanted your output to be:

```
Frank        D28
Victoria     D52
Holly        D15
Roberta      D52
```

you could use this OUTREC statement:

```
  OUTREC PARSE=(%00=(STARTAT=NONBLANK,ENDBEFR=BLANKS,FIXLEN=8),
      %01=(FIXLEN=3)),
    BUILD=(%00,14:%01)
```

Note: Multiple STARTAFT/STARTAT strings, BLANKS or NONBLANK can be used for a %nn parsed field to search for more than one string, blanks or a nonblank. The first search satisfied is used.

Example

If your input is:

```
  12   MAXCC=08
58 MINCC=00
         MAXCC=04
```

and you wanted your output to be:

```
MAXCC=08
MINCC=00
MAXCC=04
```

you could use this OUTREC statement:

```
  OUTREC PARSE=(%00=(STARTAT=C'MAXCC=',STARTAT=C'MINCC=',
                FIXLEN=8)),BUILD=(%00)
```

ENDBEFR=string

Data is to be extracted for this parsed field ending before the first byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field up to the byte before the start of the string, and the Start Pointer is set to the byte after the end of the string.

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS Application Program Guide* for details of coding character and hexadecimal string constants.

Example

If your input is:

```
Morgan
Hill;California;5000:
San Jose;California;2000:
Austin;Texas;8000:
```

and you wanted your output to be:

```
Morgan Hill   California  5000
San Jose      California  2000
Austin        Texas       8000
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(ENDBEFR=C';',FIXLEN=14),
      %01=(ENDBEFR=C';',FIXLEN=12),
      %02=(ENDBEFR=C':',FIXLEN=4)),
    BUILD=(%00,%01,%02)
```

ENDBEFR=X'00' can be used to extract null-terminated strings to fixed parsed fields.

Example

If your input contains null-terminated strings which look like this in hex:

```
F0F5F800
F1F2F3F4F500
F9F7F2F000
F500
F0F0F3F700
```

and you wanted to convert the null-terminated strings to right-aligned numeric values like this:

```
   58
12345
 9720
    5
   37
```

you could use this INREC statement:

```
  INREC PARSE=(%01=(ENDBEFR=X'00',FIXLEN=5)),
    BUILD=(%01,UFF,EDIT=(IIIIT))
```

ENDBEFR=BLANKS

Data is to be extracted for this parsed field ending before the first blank. The search for a blank begins at the Start Pointer. If a blank is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field up to the byte before the blank, and the Start Pointer is set to the first nonblank.

Example

If your input is:

```
1     Frank     D28 123   No
2 Loretta            D52   58 Yes
```

and you wanted your output to be:

```
0123 Frank   D28
0058 Loretta D52
```

you could use this OUTREC statement:

```
   OUTREC PARSE=(%00=(STARTAFT=BLANKS,ENDBEFR=BLANKS,FIXLEN=8),
      %01=(ENDBEFR=BLANKS,FIXLEN=3),
      %02=(ENDBEFR=C' ',FIXLEN=4)),
    BUILD=(%02,UFF,EDIT=(TTTT),X,%00,%01)
```

ENDAT=string

Data is to be extracted for this parsed field ending at the last byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field up to the last byte of the string, and the Start Pointer is set to the byte after the end of the string

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE Control Statement" in *z/OS Application Program Guide* for details of coding character and hexadecimal string constants.

Example

If your input is:

```
12,(June)
5852,(Gracie)
```

and you wanted your output to be:

```
(June)
(Gracie)
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAT=C'(',ENDAT=C')',FIXLEN=12)),
    BUILD=(%00)
```

ENDAT=BLANKS

Data is to be extracted for this parsed field ending at the last blank before a nonblank. The search for a blank begins at the Start Pointer. If a blank is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field up to the byte before the first nonblank, and the Start Pointer is set to the first nonblank..

Note: Multiple ENDBEFR/ENDAT strings or BLANKS can be used for a %nn parsed field to search for more than one string or blanks. The first search satisfied is used. *Example*

If your input is:

```
  12;
58:
      04/
```

and you wanted your output to be:

```
12
58
04
```

you could use this OUTREC statement:

```
  OUTREC PARSE=(%01=(ENDBEFR=C';',ENDBEFR=C':',ENDBEFR=C'/',
                FIXLEN=10)),
     BUILD=(%01,UFF,TO=ZD,LENGTH=2,80:X)
```

PAIR=APOST

Do not search for strings or blanks between apostrophe (') pairs. Use POST=APOST when you might have strings you are searching for within literals that should not satisfy the search.

Once an apostrophe is found, searching is discontinued until another apostrophe is found. Searching then continues after the second apostrophe of the pair. If an unpaired apostrophe is found, strings or BLANKS will not be recognized from that point on, so be careful not to set the Start Pointer in the middle of an apostrophe pair.

Do not specify PAIR=APOST if you want to search for a string containing an apostrophe, because the string will not be searched for within the paired apostrophes. *Example*

If your input is:

```
'23','12,567,823','5,032'
'9,903','18,321','8'
```

and you wanted your output to be:

```
        23 12,567,823      5,032
     9,903     18,321          8
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(ENDBEFR=C',',FIXLEN=12,PAIR=APOST),
                %01=(ENDBEFR=C',',FIXLEN=12,PAIR=APOST),
                %02=(FIXLEN=12)),
         BUILD=(%00,UFF,EDIT=(II,III,IIT),X,
                %01,UFF,EDIT=(II,III,IIT),X,
                %02,UFF,EDIT=(II,III,IIT))
```

With PAIR=APOST, the commas outside the apostrophe pairs satisfy the search, but the commas within the apostrophe pairs do not satisfy the search.

PAIR=QUOTE

Do not search for strings or blanks between quote (") pairs. Use POST=QUOTE when you might have strings you are searching for within literals that should not satisfy the search.

Once a quote is found, searching is discontinued until another quote is found. Searching then continues after the second quote of the pair. If an unpaired quote is found, strings or BLANKS will not be recognized from that point on, so be careful not to set the Start Pointer in the middle of a quote pair.

Do not specify PAIR=QUOTE if you want to search for a string containing a quote, because the string will not be searched for within the paired quotes. *Example*

If your input is:

```
"23","12,567,823","5,032"
"9,903","18,321","8"
```

and you wanted your output to be:

```
       23 12,567,823      5,032
    9,903     18,321          8
```

you could use this INREC statement:

```
INREC PARSE=(%00=(ENDBEFR=C',',FIXLEN=12,PAIR=QUOTE),
             %01=(ENDBEFR=C',',FIXLEN=12,PAIR=QUOTE),
             %02=(FIXLEN=12)),
      BUILD=(%00,UFF,EDIT=(II,III,IIT),X,
             %01,UFF,EDIT=(II,III,IIT),X,
             %02,UFF,EDIT=(II,III,IIT))
```

With PAIR=QUOTE, the commas outside the quote pairs satisfy the search, but the commas within the quote pairs do not satisfy the search.

Default for PARSE: None; must be specified.

PARSE in IFTHEN clauses

- WHEN=INIT: PARSE=(definitions) defines %nn fixed parsed fields into which variable position/length fields are extracted for all records. You can use the %nn parsed fields defined in a WHEN=INIT clause in the BUILD or OVERLAY operand of that clause and all subsequent IFTHEN clauses.

  Sample Syntax:

  ```
  INREC IFOUTLEN=50,
        IFTHEN=(WHEN=INIT,
         PARSE=(%01=(ABSPOS=11,STARTAT=NONBLANK,
                     ENDBEFR=C',',FIXLEN=12),
                %02=(ENDBEFR=C',',FIXLEN=10),
                %03=(FIXLEN=12))),
        IFTHEN=(WHEN=(5,2,CH,EQ,C'US'),BUILD=(%02,%03)),
        IFTHEN=(WHEN=(5,2,CH,EQ,C'UK'),BUILD=(%01,%02)),
        IFTHEN=(WHEN=NONE,BUILD=(%03,%01))
  ```

- WHEN=(logexp): PARSE=(definitions) defines %nn fixed parsed fields into which variable position/length fields are extracted for each record for which the logical expression is true. You can only use the %nn parsed fields defined in a WHEN=(logexp) clause in the BUILD or OVERLAY operand of that WHEN=(logexp) clause.

  Sample Syntax:

  ```
  OUTREC IFTHEN=(WHEN=(45,2,CH,EQ,C'AL'),
          PARSE=(%01=(ENDBEFR=C',',FIXLEN=12),
                 %02=(FIXLEN=10)),
          OVERLAY=(21:%02,51:%01))
  ```

- WHEN=ANY: PARSE=(definitions) defines %nn fixed parsed fields into which variable position/length fields are extracted for each record for which the logical expression in any WHEN=(logexp) clause is true. You can only use the %nn parsed fields defined in a WHEN=ANY clause in the BUILD or OVERLAY operand of that WHEN=ANY clause.
- WHEN=NONE: PARSE=(definitions) defines %nn fixed parsed fields into which variable position/length fields are extracted for each record for which no logical expression was true. You can only use the %nn parsed fields defined in a WHEN=NONE clause in the BUILD or OVERLAY operand of that WHEN=NONE clause.

  Sample Syntax:

```
OUTFIL IFTHEN=(WHEN=(5,2,ZD,GT,+5),
               PARSE=(%01=(STARTAT=C'<',ENDAT=C'>',FIXLEN=12),
                      %02=(STARTAFT=BLANKS,FIXLEN=10)),
               BUILD=(5,2,21:%01,X,%02,HEX)),
              IFTHEN=(WHEN=NONE,
               PARSE=(%03=(STARTAFT=C'(',ENDBEFR=C')',FIXLEN=8)),
               BUILD=(5,2,%03,SFF,M12))
```

Example 1

```
INREC PARSE=(%00=(ENDBEFR=C',',FIXLEN=11),
             %01=(ENDBEFR=C',',FIXLEN=5),
             %02=(FIXLEN=6)),
      OVERLAY=(31:%00,42:%01,47:%02)
SORT FIELDS=(31,11,CH,A,42,5,UFF,A,47,6,SFF,D)
OUTREC BUILD=(1,30)
```

This example illustrates how you can sort FB input records with variable position/length fields, such as comma separated values.

The 30-byte input records might look like this:

```
Marketing,96218,+27365
Development,3807,+1275
Research,7283,+5001
Development,1700,-5316
Research,978,+13562
Development,3807,-158
Research,7283,+5002
Marketing,52,-8736
Development,5781,+2736
Marketing,52,+1603
Research,16072,-2022
```

We want to sort the first field as character ascending, the second field as unsigned numeric ascending and the third field as signed numeric descending.

Note that each record has three variable fields in comma separated value format. The fields do not start and end in the same position in every record and do not have the same length in every record. The first and second fields end with a comma and the third field ends with a blank.

In order to sort variable fields like these, we use the PARSE and OVERLAY functions of INREC to create a fixed parsed copy of each variable field. We use %00 to create an 11-byte fixed parsed field into which we extract the value before the first comma. We use %01 to create a 5-byte fixed parsed field into which we extract the value after the first comma and before the second comma. We use %02 to create a 6-byte fixed parsed field into which

we extract the value after the second comma. Then we SORT on the fixed parsed fields. Finally, we use OUTREC to remove the fixed parsed fields.

After the INREC statement is processed, the records look like this:

```
Marketing,96218,+27365        Marketing  96218+27365
Development,3807,+1275        Development3807 +1275
Research,7283,+5001           Research   7283 +5001
Development,1700,-5316        Development1700 -5316
Research,978,+13562           Research   978  +13562
Development,3807,-158         Development3807 -158
Research,7283,+5002           Research   7283 +5002
Marketing,52,-8736            Marketing  52   -8736
Development,5781,+2736        Development5781 +2736
Marketing,52,+1603            Marketing  52   +1603
Research,16072,-2022          Research   16072-2022
```

Since the second fixed parsed field is unsigned and left-justified, we sort it with the UFF format. Since the the third fixed parsed field is signed and left-justified, we sort it with the SFF format.

After the SORT and OUTREC statements are processed, the output records look like this:

```
Development,1700,-5316
Development,3807,+1275
Development,3807,-158
Development,5781,+2736
Marketing,52,+1603
Marketing,52,-8736
Marketing,96218,+27365
Research,978,+13562
Research,7283,+5002
Research,7283,+5001
Research,16072,-2022
```

Example 2

```
INREC PARSE=(%00=(ENDBEFR=C',',FIXLEN=11),
             %01=(ENDBEFR=C',',FIXLEN=5),
             %02=(FIXLEN=6)),
      BUILD=(1,4,5:%00,16:%01,21:%02,27:5)
SORT FIELDS=(5,11,CH,A,16,5,UFF,A,21,6,SFF,D)
OUTREC BUILD=(1,4,27)
```

This example illustrates how you can sort VB input records with variable position/length fields, such as comma separated values. This example is very similar to the previous example for FB records, except that with VB records we need to copy the fixed parsed fields after the 4-byte RDW rather than at the end of the records.

The VB input records might look like this:

```
Length|Data
    26|Marketing,96218,+27365
    26|Development,3807,+1275
    23|Research,7283,+5001
    26|Development,1700,-5316
    23|Research,978,+13562
    25|Development,3807,-158
    23|Research,7283,+5002
    22|Marketing,52,-8736
    26|Development,5781,+2736
    22|Marketing,52,+1603
    24|Research,16072,-2022
```

After the INREC statement is processed, the records look like this:

```
Length|Data
    48|Marketing  96218+27365Marketing,96218,+27365
    48|Development3807 +1275 Development,3807,+1275
    45|Research   7283 +5001 Research,7283,+5001
    48|Development1700 -5316 Development,1700,-5316
    45|Research   978  +13562Research,978,+13562
    47|Development3807 -158  Development,3807,-158
    45|Research   7283 +5002 Research,7283,+5002
    44|Marketing  52   -8736 Marketing,52,-8736
    48|Development5781 +2736 Development,5781,+2736
    44|Marketing  52   +1603 Marketing,52,+1603
    46|Research   16072-2022 Research,16072,-2022
```

After the SORT and OUTREC statements are processed, the output records look like this:

```
Length|Data
    26|Development,1700,-5316
    26|Development,3807,+1275
    25|Development,3807,-158
    26|Development,5781,+2736
    22|Marketing,52,+1603
    22|Marketing,52,-8736
    26|Marketing,96218,+27365
    23|Research,978,+13562
    23|Research,7283,+5002
    23|Research,7283,+5001
    24|Research,16072,-2022
```

Example 3

```
  OPTION COPY
  OUTFIL HEADER2=(1:'First',13:'Initial',22:'Last',37:'Score',/,
                  1:10'-',13:7'-',22:11'-',35:7'-'),
     PARSE=(%00=(STARTAFT=C'LAST-> ',ENDBEFR=C' ',FIXLEN=11),
            %01=(STARTAFT=C'FIRST-> ',ENDBEFR=C' ',FIXLEN=11),
            %02=(STARTAFT=C'INITIAL-> ',ENDBEFR=C' ',FIXLEN=7),
            %03=(STARTAFT=C'SCORE-> ',FIXLEN=7)),
     BUILD=(1:%01,13:%02,22:%00,
            35:%03,SFF,EDIT=(SIIIT.T),SIGNS=(,-))
```

This example illustrates how you can create a report from FB input records with variable position/length fields, such as keyword delimited values.

The 70-byte input records might look like this:

```
     LAST-> Clark   FIRST-> Oscar     INITIAL-> D   SCORE-> +98.2
  LAST-> Roberts  FIRST-> Harriet   INITIAL->    SCORE-> -152.6
   LAST-> Stein FIRST-> Gertrude INITIAL-> V SCORE-> +5.1
```

Note that each record has four variable fields each identified by a specific keyword. The fields do not start and end in the same position in every record and they have different lengths in different records.

The output report has RECFM=FBA and LRECL=42 and looks like this:

```
First       Initial  Last           Score
----------  -------  -----------  -------
Oscar       D        Clark          98.2
Harriet              Roberts      -152.6
Gertrude    V        Stein            5.1
```

In order to create a report like this from variable position/length fields, we use HEADER2 to create the page header, and PARSE and BUILD to create a fixed parsed field from each variable field. We use %00 to create an 11-byte fixed parsed field with the extracted 'LAST-> ' value. We use %01 to create an 11-byte fixed parsed field with the extracted 'FIRST-> ' value. We use %02 to create a 7-byte fixed parsed field with the extracted 'INITIAL-> ' value. We use %03 to create a 7-byte fixed parsed field with the extracted 'SCORE-> ' value. Since the fourth field (%03) is extracted as signed and padded on the right with blanks, we treat it as SFF format in order to edit it.

Note: Example 1 in "Symbols for %nn Parsed Fields (sym,%nn)" below shows this same example using DFSORT Symbols.

Example 4

```
  INREC IFTHEN=(WHEN=INIT,
     PARSE=(%1=(FIXLEN=10,STARTAFT=C'First="',ENDBEFR=C'"'))),
    IFTHEN=(WHEN=INIT,
     PARSE=(%2=(FIXLEN=10,STARTAFT=C'Middle="',ENDBEFR=C'"'))),
    IFTHEN=(WHEN=INIT,
     PARSE=(%3=(FIXLEN=10,STARTAFT=C'Last="',ENDBEFR=C'"'))),
    IFTHEN=(WHEN=INIT,
     PARSE=(%4=(FIXLEN=10,STARTAFT=C'Wife="',ENDBEFR=C'"')),
     BUILD=(%1,13:%2,25:%3,37:%4))
  SORT FIELDS=(25,10,CH,A,1,10,CH,A,13,10,CH,A)
  OUTFIL HEADER2=('First',13:'Middle',
      25:C'Last',37:'Wife',/,10C'-',13:10C'-',
      25:10C'-',37:10C'-')
```

This example illustrates how you can create a sorted report from FB input records with keyword values that can occur in any order or not occur at all.

The 80-byte input records might look like this:

```
   Last="Buchanan" First="James"
 Wife="Louisa" First="John" Middle="Quincy" Last="Adams"
  First="George" Last="Washington" Wife="Martha"
Last="Clinton" Wife="Hillary" Middle="Jefferson" First="William"
 First="John" Wife="Abigail" Last="Adams"
```

Note that each record has up to four variable fields each identified by a specific keyword (First, Middle, Last, Wife). In each record, the fields can be in any order, do not start and end in the same position and can have different lengths.

We want to sort the records by last name, first name and middle name, and create a report with fixed headings and values for the first name, middle name, last name and wife's name that looks like this:

```
First       Middle      Last        Wife
----------  ----------  ----------  ----------
John                    Adams       Abigail
John        Quincy      Adams       Louisa
James                   Buchanan
William     Jefferson   Clinton     Hillary
George                  Washington  Martha
```

In order to extract and sort variable fields like these, we use a separate IFTHEN PARSE clause for each keyword. This allows us to find a particular keyword anywhere in the record or ignore it if it is not in the record. If we used PARSE instead of IFTHEN PARSE, a missing keyword would cause any keywords that followed to be ignored. But because each IFTHEN PARSE starts scanning at position 1 by default, we can look for each keyword independently of the others. If a keyword is missing, the parsed field is set to all blanks (for example, %2 and %4 are set to blanks for the James Buchanan record).

We use %1 to create a 10-byte fixed parsed field into which we extract the first name. We use %2 to create a 10-byte fixed parsed field into which we extract the middle name. We use %3 to create a 10-byte fixed parsed field into which we extract the last name. We use %4 to create a 10-byte fixed parsed field into which we extract the wife's name. We use BUILD to reformat the records to contain the fixed parsed fields. Then we SORT on the fixed parsed fields. Finally, we use OUTFIL to create the headings.

Example 5

```
  OPTION COPY
  OUTREC PARSE=(%01=(ABSPOS=2,FIXLEN=13,ENDBEFR=C'",'),
                %=(ENDBEFR=C'",'),
                %03=(FIXLEN=6,ENDBEFR=C'",'),
                %04=(FIXLEN=6,ENDBEFR=C'",'),
                %05=(FIXLEN=12,ENDBEFR=C'")),
    BUILD=(%01,20:%03,SFF,ADD,%04,SFF,EDIT=(SIIT.T),SIGNS=(,-),
           31:%05)
```

This example illustrates how you can reformat records containing variable position/length fields, such as comma separated values.

The 80-byte input records might look like this:

```
"Buffy Summers","F","+725.8","-27.3","Sunnydale"
"Bruce Wayne","M","-5.3","-173.2","Gotham City"
"Clark Kent","M","+21.3","-15.8","Metropolis"
"Diana Prince","F","-16.4","+128.9","Gateway City"
```

Note that each record has five variable fields, each enclosed in quotes and separated by a comma. The fields do not start and end in the same position in every record and have different lengths in different records.

The 42-byte output records should look like this:

```
Buffy Summers       698.5     Sunnydale
Bruce Wayne        -178.5     Gotham City
Clark Kent            5.5     Metropolis
Diana Prince        112.5     Gateway City
```

The first fixed-length output field corresponds to the first variable input field. The second fixed-length output field corresponds to the total of the third and fourth variable input fields. The third fixed-length output field corresponds to the fifth variable input field.

In order to reformat the input records for output, we use PARSE and BUILD to create the fixed parsed fields we need from the variable position/length fields; the quotes around each value are removed. %00 is used for the first input field. % is used to ignore the second input field. %03 and %04 are used for the third and fourth input fields which are added together using SFF format. %05 is used for the fifth input field.

Example 6

```
OPTION COPY
OUTREC IFTHEN=(WHEN=INIT,
         PARSE=(%00=(FIXLEN=3,ENDBEFR=C'.'),
                %01=(FIXLEN=3,ENDBEFR=C'.'),
                %02=(FIXLEN=3,ENDBEFR=C'.'),
                %03=(FIXLEN=3)),
         BUILD=(%00,4:C'.',5:%01,8:C'.',9:%02,12:C'.',13:%03)),
   IFTHEN=(WHEN=(1,3,CH,EQ,C'167',AND,13,3,CH,EQ,C'99'),
           OVERLAY=(5:C'113',9:C'75 ',1:1,15,SQZ=(SHIFT=LEFT))),
   IFTHEN=(WHEN=NONE,
           BUILD=(1:1,15,SQZ=(SHIFT=LEFT)))
```

This example illustrates how you can modify variable position/length fields, such as ftp addresses.

The 15-byte input records might look like this:

```
167.113.117.99
167.90.18.99
167.80.118.98
165.250.89.562
167.125.890.95
168.250.89.99
167.125.890.99
0.0.0.0
167.90.580.99
```

We want to convert each FTP address of the form 167.x.y.99 to 167.113.75.99. x and y can be any 1-3 digit value. Thus, the 15-byte output records should look like this:

```
167.113.75.99
167.113.75.99
167.80.118.98
165.250.89.562
167.125.890.95
168.250.89.99
167.113.75.99
0.0.0.0
167.113.75.99
```

In order to reformat the input records for output, we use IFTHEN clauses as follows:

- IFTHEN WHEN=INIT clause: We PARSE the four variable numeric values into four 3-byte fixed parsed fields using %00, %01, %02 and %03 respectively. We reformat the record with %00, a period, %01, a period, %02, a period and %03. At this point, the reformatted records look like this:

```
167.113.117.99
167.90 .18 .99
167.80 .118.98
165.250.89 .562
167.125.890.95
168.250.89 .99
167.125.890.99
0  .0  .0  .0
167.90 .580.99
```

- IFTHEN WHEN=(logexp) clause: If the first fixed-length field is '167' and the fourth fixed-length field is '99 ', we overlay the second fixed-length field with '113' and the third fixed-length field with '75 '. Then we squeeze the fields to the left to remove the blanks.
- IFTHEN WHEN=NONE clause: If the first fixed-length field is not '167' or the fourth fixed-length field is not '99 ', we squeeze the fields to the left to remove the blanks.

Symbols for %nn Parsed Fields (sym,%nn)

Introduction

A symbol can now be used for a %nn parsed field. For example, if Account,%01 is defined in SYMNAMES, Account can be used for %01. A symbol for %nn can be used in the PARSE, BUILD, OVERLAY, IFTHEN PARSE, IFTHEN BUILD and IFTHEN OVERLAY operands of the INREC, OUTREC and OUTFIL statements wherever %nn can be used. A symbol for %nn results in substitution of %nn.

Detailed Description and Syntax

A symbol for a parsed field is specified as follows:

```
symbol,%nn
```

A parsed field can be specified as %nn with nn as 00-99 or as %n with n as 0-9. %0n will be substituted for %n. A symbol for a parsed field must be used only where such a field is allowed and has the desired result. Otherwise, substitution of %nn for the symbol will result in an error message. For example, if the following SYMNAMES statement is specified:

```
Revenue,%03
```

Revenue can be used in an OUTREC statement such as:

```
OUTREC PARSE=(Revenue=(STARTAT=C'+',FIXLEN=9)),
       OVERLAY=(31:Revenue,UFF,EDIT=(III,IIT,TTT))
```

because a parsed field is allowed in the PARSE operand and the OVERLAY operand. However, Revenue will result in an error message if used in a SORT statement such as:

```
SORT FIELDS=(Revenue,UFF,A)
```

because a parsed field is not allowed in a SORT statement.

Make sure the parsed fields that will be substituted for your symbols are appropriate.

As an example of how parsed fields can be used, if you specify the following SYMNAMES statements:

```
branch,1,7
tab,X'05'
comma,','
first_name,%00
last_name,%1
date,%02
company,%03
city,%04
state,%05
```

SYMNOUT will show the following symbol table:

```
branch,1,7
tab,X'05'
comma,C','
first_name,%00
last_name,%01
date,%02
company,%03
city,%04
state,%05
```

This DFSORT control statement:

```
  INREC PARSE=(first_name=(ABSPOS=9,FIXLEN=12,ENDBEFR=comma),
         last_name=(FIXLEN=15,ENDBEFR=comma),
         %=(ENDBEFR=tab),
         date=(FIXLEN=10,ENDBEFR=tab),
         company=(FIXLEN=31,ENDBEFR=tab),
         city=(FIXLEN=14,ENDBEFR=tab),
         state=(FIXLEN=12)),
    BUILD=(branch,9:first_name,22:last_name,
           38:date,JFY=(SHIFT=RIGHT),
           49:company,83:city,98:state)
```

will be transformed to:

```
 INREC PARSE=(%00=(ABSPOS=9,FIXLEN=12,ENDBEFR=C','),%01=(FIXLEN=15,ENDB*
               EFR=C','),%=(ENDBEFR=X'05'),%02=(FIXLEN=10,ENDBEFR=X'05'*
               ),%03=(FIXLEN=31,ENDBEFR=X'05'),%04=(FIXLEN=14,ENDBEFR=X*
               '05'),%05=(FIXLEN=12)),BUILD=(1,7,9:%00,22:%01,38:%02,JF*
               Y=(SHIFT=RIGHT),49:%03,83:%04,98:%05)
```

Example 1

```
...
//SYMNAMES DD *
Last,%00
First,%01
Init,%02
Score,%03
KW_Last,'LAST-> '
KW_First,'FIRST-> '
KW_Init,'INITIAL-> '
KW_Score,'SCORE-> '
HD_First,'First'
HD_Init,'Initial'
HD_Last,'Last'
HD_Score,'Score'
/*
//SYSIN DD *
  OPTION COPY
  OUTFIL HEADER2=(1:HD_First,13:HD_Init,22:HD_Last,37:HD_Score,/,
                  1:10'-',13:7'-',22:11'-',35:7'-'),
     PARSE=(Last=(STARTAFT=KW_Last,ENDBEFR=C' ',FIXLEN=11),
            First=(STARTAFT=KW_First,ENDBEFR=C' ',FIXLEN=11),
            Init=(STARTAFT=KW_Init,ENDBEFR=C' ',FIXLEN=7),
            Score=(STARTAFT=KW_Score,FIXLEN=7)),
     BUILD=(1:First,13:Init,22:Last,
            35:Score,SFF,EDIT=(SIIIT.T),SIGNS=(,-))
/*
```

This example shows how you can use Symbols for Example 3 from "Parsing Variable Fields (PARSE, %nn)" above.

Comparing Past and Future Date Constants (DATEn-r, DATEn+r)

Introduction

The COND operand of the INCLUDE and OMIT statements, the INCLUDE and OMIT operands of the OUTFIL statement, and the IFTHEN WHEN operand of the INREC, OUTREC and OUTFIL statements now allow you to compare date fields in various formats to past and future dates (relative to the date of your DFSORT run). You can set up the past and future dates you need using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r, DATEnP-r, Y'DATEn'+r and Y'DATEn'-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and &DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

Decimal constants, character constants and Y-constants for past and future dates can be compared to appropriate fields in the same way that constants for current dates can be compared to appropriate fields. See "INCLUDE Control Statement" in *z/OS DFSORT Application Programming Guide* for details of valid field-to-constant comparisons.

Past Date as Decimal Number

DATE1P-d, &DATE1P-d, DATE2P-m, &DATE2P-m, DATE3P-d or &DATE3P-d can be used to generate a decimal number for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

The table below shows the form of the decimal number constant generated for each past date operand along with an example of the actual decimal number generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year (001-366).

Decimal Numbers for Past Dates

Format of Operand	Format of Constant	Example of Operand	Example of Constant
DATE1P-d	+yyyymmdd	DATE1P-30	+20050522
DATE2P-m	+yyyymm	DATE2P-12	+200406
DATE3P-d	+yyyyddd	DATE3P-172	+2004366

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
INCLUDE COND=(21,7,ZD,GE,DATE3P-30)
```

Future Date as Decimal Number

DATE1P+d, &DATE1P+d, DATE2P+m, &DATE2P+m, DATE3P+d or &DATE3P+d can be used to generate a decimal number for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

The table below shows the form of the decimal number constant generated for each future date operand along with an example of the actual decimal number generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year (001-366).

Decimal Numbers for Future Dates

Format of Operand	Format of Constant	Example of Operand	Example of Constant
DATE1P+d	+yyyymmdd	DATE1P+11	+20050702
DATE2P+m	+yyyymm	DATE2P+2	+200508
DATE3P+d	+yyyyddd	DATE3P+200	+2006007

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
OUTFIL OMIT=(18,5,PD,LT,DATE1P+1)
```

Past Date as Character String

DATE1-d, &DATE1-d, DATE1(c)-d, &DATE1(c)-d, DATE2-m, &DATE2-m, DATE2(c)-m, &DATE2(c)-m, DATE3-d, &DATE3-d, DATE3(c)-d or &DATE3(c)-d can be used to generate a character string for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

The table below shows the form of the character string constant generated for each past date operand along with an example of the actual character string generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character except a blank.

Character Strings for Past Dates

Format of Operand	Format of Constant	Example of Operand	Example of Constant
DATE1-d	C'yyyymmdd'	DATE1-1	C'20050620'
DATE1(c)-d	C'yyyycmmcdd'	DATE1(-)-60	C'2005-04-22'
DATE2-m	C'yyyymm'	DATE2-6	C'200412'
DATE2(c)-m	C'yyyycmm'	DATE2(/)-1	C'2005/05'
DATE3-d	C'yyyyddd'	DATE3-300	C'2004238'
DATE3(c)-d	C'yyyycddd'	DATE3(.)-21	C'2005.151'

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
INREC IFTHEN=(WHEN=(31,6,CH,EQ,DATE2(/)-12),
         OVERLAY=(52:C'Y'))
```

Future Date as Character String

DATE1+d, &DATE1+d, DATE1(c)+d, &DATE1(c)+d, DATE2+m, &DATE2+m, DATE2(c)+m, &DATE2(c)+m, DATE3+d, &DATE3+d, DATE3(c)+d or &DATE3(c)+d can be used to generate a character string for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

The table below shows the form of the character string constant generated for each future date operand along with an example of the actual character string generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character except a blank.

Character Strings for Future Dates

Format of Operand	Format of Constant	Example of Operand	Example of Constant
DATE1+d	C'yyyymmdd'	DATE1+11	C'20050702'
DATE1(c)+d	C'yyyycmmcdd'	DATE1(/)+90	C'2005/09/19'
DATE2+m	C'yyyymm'	DATE2+2	C'200508'
DATE2(c)+m	C'yyyycmm'	DATE2(.)+25	C'2007.07'
DATE3+d	C'yyyyddd'	DATE3+200	C'2006007'
DATE3(c)+d	C'yyyycddd'	DATE3(-)+1	C'2005-171'

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
OMIT COND=(21,7,CH,NE,DATE3+200)
```

Past Date as Y-Constant

Y'DATE1'-d, Y'DATE2'-m and Y'DATE3'-d can be used to generate a Y-constant for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

The table below shows the form of the Y-constant generated for each past date operand along with an example of the actual Y-constant generated when the date of the run is June 21, 2005. yy represents the two-digit year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year.

Y-Constants for Past Dates

Format of Operand	Format of Constant	Example of Operand	Example of Constant
Y'DATE1'-d	Y'yymmdd'	Y'DATE1'-30	Y'050522'
Y'DATE2'-m	Y'yymm'	Y'DATE2'-12	Y'0406'
Y'DATE3'-d	Y'yyddd'	Y'DATE3'-172	Y'04366'

Sample Syntax

```
INCLUDE COND=(3,4,Y2T,GE,Y'DATE2'-3)
```

Future Date as Y-Constant

Y'DATE1'+d, Y'DATE2'+m and Y'DATE3'+d can be used to generate a Y-constant for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

The table below shows the form of Y-constant generated for each future date operand along with an example of the actual Y-constant generated when the date of the run is June 21, 2005. yy represents the two-digit year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year.

Y-Constants for Future Dates

Format of Operand	Format of Constant	Example of Operand	Example of Constant
Y'DATE1'+d	Y'yymmdd'	Y'DATE1'+11	Y'050702'
Y'DATE2'+m	Y'yymm'	Y'DATE2'+2	Y'0508'
Y'DATE3'+d	Y'yyddd'	Y'DATE3'+200	Y'06007'

Sample Syntax

```
OUTFIL OMIT=(21,6,Y2W,EQ,Y'DATE1'+60)
```

Example 1

```
INCLUDE COND=(15,7,CH,GE,DATE3-7,AND,15,7,CH,LE,DATE3+7)
```

This example illustrates how to include records in which a character date of the form C'yyyyddd' in bytes 15-21 is between 7 days in the past and 7 days in the future, relative to the current date. DATE3-7 generates a character constant in the form C'yyyyddd' where yyyyddd is the current date minus 7 days. DATE3+7 generates a character constant in the form C'yyyyddd' where yyyyddd is the current date plus 7 days.

Example 2

```
OUTFIL OMIT=(21,10,CH,GE,DATE1(-)-365)
```

This example illustrates how to omit records in which a character date of the form C'yyyy-mm-dd' in bytes 21-30 is within 365 days of the current date. DATE1(-)-365 generates a character constant in the form C'yyyy-mm-dd' where yyyymmdd is the current date minus 365 days.

Inserting Past and Future Date Constants (DATEn-r, DATEn+r)

Introduction

The BUILD, OVERLAY, IFTHEN BUILD and IFTHEN OVERLAY operands of the INREC, OUTREC and OUTFIL statements now allow you to insert past and future dates (relative to the date of your DFSORT run) into your records in various forms. You can insert the past and future dates you need using new DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r constants. &DATEn+r, &DATEn-r, &DATEn(c)+r, &DATEn(c)-r, &DATEnP+r and &DATEnP-r can be used as aliases for DATEn+r, DATEn-r, DATEn(c)+r, DATEn(c)-r, DATEnP+r and DATEnP-r, respectively.

Packed decimal and character constants for past and future dates can be inserted in your reformatted records in the same way that other packed decimal and character constants can be inserted.

Past Date as Packed Decimal Constant

DATE1P-d, &DATE1P-d, DATE2P-m, &DATE2P-m, DATE3P-d or &DATE3P-d can be used to generate a packed decimal constant for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

The table below shows the form of the packed decimal constant generated for each past date operand along with an example of the actual constant generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year (001-366).

Packed Decimal Constants for Past Dates

Format of Operand	Format of Constant	Length (bytes)	Example of Operand	Example of Constant
DATE1P-d	P'yyyymmdd'	5	DATE1P-30	P'20050522'
DATE2P-m	P'yyyymm'	4	DATE2P-12	P'200406'
DATE3P-d	P'yyyyddd'	4	DATE3P-172	P'2004366'

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
OUTREC OVERLAY=(115:DATE2P(.)-12)
```

Future Date as Packed Decimal Constant

DATE1P+d, &DATE1P+d, DATE2P+m, &DATE2P+m, DATE3P+d or &DATE3P+d can be used to generate a packed decimal constant for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

The table below shows the form of the packed decimal constant generated for each future date operand along with an example of the actual constant generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year (001-366).

Packed Decimal Constants for Future Dates

Format of Operand	Format of Constant	Length (bytes)	Example of Operand	Example of Constant
DATE1P+d	P'yyyymmdd'	5	DATE1P+11	P'20050702'
DATE2P+m	P'yyyymm'	4	DATE2P+2	P'200508'
DATE3P+d	P'yyyyddd'	4	DATE3P+200	P'2006007'

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
OUTFIL BUILD=(1,40,DATE1P+1,/,
              1,40,DATE2P+1,/,
              1,40,DATE3P+1)
```

Past Date as Character String

DATE1-d, &DATE1-d, DATE1(c)-d, &DATE1(c)-d, DATE2-m, &DATE2-m, DATE2(c)-m, &DATE2(c)-m, DATE3-d, &DATE3-d, DATE3(c)-d or &DATE3(c)-d can be used to generate a character string for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

The table below shows the form of the character string constant generated for each past date operand along with an example of the actual character string generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character except a blank.

Character Strings for Past Dates

Format of Operand	Format of Constant	Length (bytes)	Example of Operand	Example of Constant
DATE1-d	C'yyyymmdd'	8	DATE1-1	C'20050620'
DATE1(c)-d	C'yyyycmmcdd'	10	DATE1(-)-60	C'2005-04-22'
DATE2-m	C'yyyymm'	6	DATE2-6	C'200412'
DATE2(c)-m	C'yyyycmm'	7	DATE2(/)-1	C'2005/05'
DATE3-d	C'yyyyddd'	7	DATE3-300	C'2004238'
DATE3(c)-d	C'yyyycddd'	8	DATE3(.)-21	C'2005.151'

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
INREC BUILD=(5X,C'*',DATE3-30,C'*',1,30)
```

Future Date as Character String

DATE1+d, &DATE1+d, DATE1(c)+d, &DATE1(c)+d, DATE2+m, &DATE2+m, DATE2(c)+m, &DATE2(c)+m, DATE3+d, &DATE3+d, DATE3(c)+d or &DATE3(c)+d can be used to generate a character string for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

The table below shows the form of the character string constant generated for each future date operand along with an example of the actual character string generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character except a blank.

Character Strings for Future Dates

Format of Operand	Format of Constant	Length (bytes)	Example of Operand	Example of Constant
DATE1+d	C'yyyymmdd'	8	DATE1+11	C'20050702'
DATE1(c)+d	C'yyyycmmcdd'	10	DATE1(/)+90	C'2005/09/19'
DATE2+m	C'yyyymm'	6	DATE2+2	C'200508'
DATE2(c)+m	C'yyyycmm'	7	DATE2(.)+25	C'2007.07'
DATE3+d	C'yyyyddd'	7	DATE3+200	C'2006007'
DATE3(c)+d	C'yyyycddd'	8	DATE3(-)+1	C'2005-171'

Note: You can precede each of the operands in the table with an & with identical results.

Sample Syntax

```
OUTREC IFTHEN=(WHEN=(21,1,CH,EQ,C'Y'),
   OVERLAY=(52:DATE1+7))
```

Example 1

```
OUTREC OVERLAY=(51:DATE1(/)-1,X,DATE1(/)+1)
```

This example illustrates how character constants for yesterday's date and tomorrow's date can be inserted in the reformatted records. DATE1(/)-1 generates a character constant in the form C'yyyy/mm/dd' where yyyymmdd is yesterday's date. DATE1(/)+1 generates a character constant in the form C'yyyy/mm/dd' where yyyymmdd is tomorrow's date.

Numeric and Non-Numeric Tests (NUM)

Introduction

The COND operand of the INCLUDE and OMIT statements, the INCLUDE and OMIT operands of the OUTFIL statement, and the IFTHEN WHEN operand of the INREC, OUTREC and OUTFIL statements now allow you to test a field for numerics (field,EQ,NUM) or non-numerics (field,NE,NUM) in character (FS), zoned decimal (ZD) or packed decimal (PD) format.

Tutorial

Suppose you think that some of the values in the Employees field might contain invalid numeric data, and you want to select the records with those values, if any. Each byte of the 4-byte Employees field should contain '0' through '9'; you would consider any other character, such as 'A' or '.' to be invalid. '1234' is a valid numeric value; it contains all numerics. '12.3' is an invalid numeric value; it contains a non-numeric.

You can use one of the numeric test capabilities of the INCLUDE statement to collect the records you want as follows:

```
INCLUDE COND=(18,4,FS,NE,NUM)
```

If the value in the field (18,4,FS) is not equal (NE) to numerics (NUM), the record is included. The records in the output data set will be those in which the field has non-numerics (a character other than '0'-'9' appears somewhere in the field).

Use NUM to indicate a test for numerics or non-numerics.

Use EQ to test for numerics, or NE to test for non-numerics.

Use FS format for the field if you want to test for character numerics ('0'-'9' in every byte).

Use ZD format for the field if you want to test for zoned decimal numerics ('0'-'9' in all non-sign bytes; X'F0'-X'F9', X'D0'-X'D9' or X'C0'-X'C9' in the sign byte).

Use PD format for the field if you want to test for packed decimal numerics (0-9 for all digits; F, D or C for the sign).

Here's an INCLUDE statement that only includes records in which the Revenue field and Profit field have packed decimal numerics (that is, there are no invalid packed decimal values in these fields).

```
INCLUDE COND=(22,6,PD,EQ,NUM,AND,28,6,PD,EQ,NUM)
```

Detailed Description and Syntax

You can use NUM in the COND operand of the INCLUDE and OMIT statements, in the INCLUDE and OMIT operands of the OUTFIL statement and in the IFTHEN WHEN operand of the INREC, OUTREC and OUTFIL statements.

Numeric Tests

You can test a field for numerics or non-numerics in character, zoned decimal or packed decimal format.

For example, you can include only those records in which a 5-byte field contains only '0'-'9' characters (that is, character numerics). Or you can include only those records in which a 9-byte field contains invalid zoned decimal data (that is, zoned decimal non-numerics). Or you can include only those records in which a 12-byte field contains valid packed decimal data (that is, packed decimal numerics).

A field to be tested for numerics in character format looks like this in hexadecimal:

```
FdFd...Fd
```

The field is considered to be character numeric if every d is 0-9. (This is equivalent to '0'-'9' for each character.) Otherwise, the field is considered to be character non-numeric. For example, '1234', '0001' and '9999' are all considered to be character numeric, whereas 'A234', '12.3', ' 1' and '123D' are all considered to be character non-numeric.

A field to be tested for numerics in zoned decimal format looks like this in hexadecimal:

```
zdzd...sd
```

The field is considered to be zoned decimal numeric if every z is F, every d is 0-9, and s is C, D or F. Otherwise, the field is considered to be zoned decimal non-numeric. For example, '1234' (X'F1F2F3F4'), '123D' (X'F1F2F3C4') and '123M' (X'F1F2F3D4') are all considered to be zoned decimal numeric, whereas 'A234' (X'C1F2F3F4'), '12.3' (X'F1F24BF3') and '123X' (X'F1F2F3E7') are all considered to be zoned decimal non-numeric.

A field to be tested for numerics in packed decimal format looks like this in hexadecimal:

```
dddd...ds
```

The field is considered to be packed decimal numeric if every d is 0-9, and s is C, D or F. Otherwise, the field is considered to be packed decimal non-numeric. For example, X'12345C', X'12345D' and X'12345F' are all considered to be packed decimal numeric, whereas X'A2345C', X'1B345D' and X'12F45F' are all considered to be packed decimal non-numeric.

Relational Condition Format

Two formats for the relational condition can be used:

```
>>-(p1,m1,f1,--+-EQ-+--,--NUM--)-------------------------------><
               '-NE-'
```

Or, if the FORMAT=f operand is used:

```
>>-(p1,m1,--+-----+--+-EQ-+--,--NUM--)-------------------------><
            '-f1,-'  '-NE-'
```

Numeric test operators are as follows:

EQ Equal to numerics

NE Not equal to numerics (non-numerics)

p1,m1,f1:

These variables specify the field in the input record for the numeric test.

p1 specifies the first byte of the field relative to the beginning of the input record. The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All fields must start on a byte boundary, and no field can extend beyond byte 32752.

m1 specifies the length of the field. The length can be 1 to 256 bytes.

f1 specifies the type of numerics the field is to be tested for as follows:

FS tests for numerics in character format ('0'-'9' (X'F0'-X'F9') in all bytes).

Note: CSF can be used instead of FS.

ZD tests for numerics in zoned decimal format ('0'-'9' (X'F0'-X'F9') in all non-sign bytes; X'F0'-X'F9', X'D0'-X'D9' or X'C0'-X'C9' in the sign byte)

PD tests for numerics in packed decimal format (0-9 for all digits; F, D or C for the sign).

You can use p1,m1 rather than p1,m1,f1 if you use FORMAT=f to supply the format for the field.

NUM

Specifies a test for numerics or non-numerics. The condition will be true if the field is numeric when the EQ operator is specified or if the field is non-numeric when the NE operator is specified.

Example 1

```
INCLUDE COND=(1,20,FS,EQ,NUM)
```

This example illustrates how to only include records in which the field in bytes 1 through 20 contains valid character numeric data (that is, '0'-'9' in all bytes).

Example 2

```
OUTFIL INCLUDE=(21,8,ZD,NE,NUM,OR,31,5,PD,NE,NUM)
```

This example illustrates how to only include records in which the field in bytes 21 through 28 contains invalid zoned decimal data, or the field in bytes 31 through 35 contains invalid packed decimal data (that is, one of the fields is non-numeric).

Floating-Point Display (FL)

Introduction

The BUILD, OVERLAY, IFTHEN BUILD, and IFTHEN OVERLAY operands of the INREC, OUTREC and OUTFIL statements, the TRAILERx operands of the OUTFIL statement, and the ON operand of ICETOOL's DISPLAY operator now allow you to use FL format to convert 4-byte or 8-byte hexadecimal floating-point values to integer values.

Detailed Description

p,m,FL can be used in the BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY, TRAILERx and ON (DISPLAY only) operands in the same way other numeric formats (e.g. ZD, PD, etc) can be used in those operands.

The normalized or unnormalized FL (hexadecimal floating-point) value is converted to a signed integer in the range -9223372036854775808 to 9223372036854775807. The fractional part of the FL value is lost, and in some cases the signed integer may be one of a number of possible signed integers for the FL value depending on its precision. Converted values less than -9223372036854775808 are set to -9223372036854775808. Converted values greater than 9223372036854775807 are set to 9223372036854775807.

By default, an FL value is converted to a sign and 20 decimal digits when used in the BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY or TRAILERx operands, and to a sign and 31 decimal digits when used in the ON operand.

If you are not running in z/Architecture mode, specifying an FL format field results in an error message and termination.

Sample Syntax

```
OUTFIL FNAMES=OUT1,REMOVECC,
  BUILD=(1,40,41:51,4,FL,M12),
  TRAILER1=(C'Total',41:TOT=(51,4,FL,M12))
```

Example 1

```
SORT FROM(SMFIN) TO(SMF71) USING(TY71)
DISPLAY FROM(SMF71) LIST(SMF71RPT) -
  TITLE('Low impact central storage frames') -
  BREAK(15,4,CH,L'System: ') -
  HEADER('Date') ON(11,4,DT1,E'9999-99-99') -
  HEADER('Time') ON(7,4,TM1,E'99:99:99') -
  HEADER('Min Frames') ON(925,8,FL,U10) -
  HEADER('Max Frames') ON(933,8,FL,U10) -
  HEADER('Avg Frames') ON(941,8,FL,U10) -
  BLANK PAGE
```

This example shows how floating point values can be displayed as integers in a report on SMF type-71 records with a section for each system id.

The SORT operator selects SMF type-71 records that are at least 19 bytes long and sorts them by system id, date and time to the SMF71 data set. It uses the following control statements in TY71CNTL:

```
  OMIT COND=(6,1,BI,NE,+71,OR,1,2,BI,LE,+18)
  SORT FIELDS=(15,4,CH,A,11,4,PD,A,7,4,BI,A)
```

The DISPLAY operator uses the selected type-71 records in SMF71 to print, in the SMF71RPT data set:

- A title line containing the specified title and the page number
- A break title containing the specified leading string and the SMF71SID system id character values in positions 15-18
- A heading line containing the specified underlined headings
- Data lines containing:
  - The SMF71DTE date value. This SMF date value in positions 11-14 is displayed as a C'yyyy-mm-dd' value.
  - The SMF71TME time value. This SMF time value in positions 7-10 is displayed as a C'hh:mm:ss' value.
  - The SMF71CLM minimum number of low-impact central storage frames value. This floating-point value in positions 925-932 is displayed as a 10 digit integer value. (The U10 formatting item reduces the number of digits for the integer representation of the floating-point value from 20 to 10, decreasing the column width for the field.)
  - The SMF71CLX maximum number of low-impact central storage frames value. This floating-point value in positions 933-940 is displayed as a 10 digit integer value.
  - The SMF71CLA average number of low-impact central storage frames value. This floating-point value in positions 941-948 is displayed as a 10 digit integer value.

Each system id value starts on a new page and looks as follows (several sections and records are shown with illustrative values):

```
Low impact central storage frames        - 1 -

System: SYSA

      Date       Time    Min Frames    Max Frames    Avg Frames
----------   --------   -----------   -----------   -----------
2005-08-01   11:45:00        934215       1001596        963434
2005-08-01   12:00:00        971599       1004939        984437
2005-08-01   12:15:00        970192        982565        973768

Low impact central storage frames        - 2 -

System: SYSB

      Date       Time    Min Frames    Max Frames    Avg Frames
----------   --------   -----------   -----------   -----------
2005-08-01   11:45:00        947220        985444        966581
2005-08-01   12:00:00        980120       1018982        986360
2005-08-01   12:15:00        980387       1051920       1011873
```

Splitting a File Contiguously (SPLIT1R)

Introduction

SPLIT1R is a new operand of the OUTFIL statement that allows you to write contiguous groups of records in one rotation among multiple output data sets. A specified number of records is written to each output data set and extra records are written to the last output data set.

Tutorial

Suppose you have an input data set with 16 records and you want to split them as evenly as possible with 5 records in two output data sets and six records in the other output data set. If you use:

```
OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLITBY=5
```

records 1-5 are written to OUT1, records 6-10 are written to OUT2, and records 11-15 are written to OUT3. But since SPLITBY=5 starts over with the first ddname (OUT1) after writing 5 records to the last ddname (OUT3), record 16 is written to OUT1. Thus, 6 records would go to OUT1 and they would be non-contiguous (1-5 and 16).

If instead, you only want contiguous records in each output data set, you can use OUTFIL's SPLIT1R=n operand which writes n records to each output data set and then writes the remaining records to the last output data set. Thus, whereas SPLIT and SPLITBY=n rotate many times among the output data sets, SPLIT1R=n only rotates once among the output data sets, resulting in contiguous records in each output data set.

You can use the following OUTFIL statement to ensure that the records in each output data set are contiguous:

```
OUTFIL FNAMES=(OUT1,OUT2,OUT3),SPLIT1R=5
```

For your 16 input records, these records are written to OUT1:

```
Record 01
Record 02
Record 03
Record 04
Record 05
```

these records are written to OUT2:

```
Record 06
Record 07
Record 08
Record 09
Record 10
```

and these records are written to OUT3:

```
Record 11
Record 12
Record 13
Record 14
Record 15
Record 16
```

With SPLIT1R=5, 6 records go to OUT3 and they are contiguous (11-16).

Detailed Description and Syntax

You can use the SPLIT1R=n operand in the OUTFIL statement.

```
>>-SPLIT1R=n---->
```

Splits the output records n records at a time for one rotation among the data sets of this OUTFIL group until all of the output records have been written.

As an example, if SPLIT1R=10 is specified for an input data set with 35 records and an OUTFIL group with three data sets:

- the first OUTFIL data set in the group will receive records 1-10.
- the second OUTFIL data set in the group will receive records 11-20.
- the third OUTFIL data set in the group will receive records 21-35.

The records **are** contiguous in each OUTFIL data set.

The SPLIT1R parameter cannot be used with any of the following report parameters: LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

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

Sample Syntax:

```
* WRITE RECORDS 1-20 TO PIPE1, RECORDS 21-40 TO PIPE2,
* RECORDS 41-60 TO PIPE3 AND RECORDS 61-85 TO PIPE4.
   OUTFIL FNAMES=(PIPE1,PIPE2,PIPE3,PIPE4),SPLIT1R=20

* SPLIT THE INCLUDED AND REFORMATTED OUTPUT RECORDS ONCE
* CONTIGUOUSLY BETWEEN TAPE1 AND TAPE2.
   OUTFIL FNAMES=(TAPE1,TAPE2),SPLIT1R=100,
     INCLUDE=(8,2,ZD,EQ,27),OUTREC=(5X,1,75)
```

Default for SPLIT1R: None; must be specified.

Example 1

```
OPTION COPY
OUTFIL FNAMES=(A1,A2,A3),SPLIT
OUTFIL FNAMES=(B1,B2,B3),SPLITBY=25
OUTFIL FNAMES=(C1,C2,C3),SPLIT1R=25
```

This example illustrates different ways to split 77 input records among three OUTFIL data sets.

The first OUTFIL statement uses SPLIT to rotate the records among the three OUTFIL data sets one record at a time. A1 will have records 1, 4, ..., 76. A2 will have records 2, 5, ..., 77. A3 will have records 3, 6, ..., 75. A1 will have 26 records. A2 will have 26 records. A3 will have 25 records. A1, A2 and A3 will each have non-contiguous records.

The second OUTFIL statement uses SPLITBY=25 to rotate the records among the three OUTFIL data sets 25 records at a time. B1 will have records 1-25 and 76-77. B2 will have records 26-50. B3 will have records 51-75. B1 will have 27 records. B2 will have 25 records. B3 will have 25 records. B1 will have non-contiguous records. B2 and B3 will have contiguous records.

The third OUTFIL statement uses SPLIT1R=25 to rotate the records once among the three OUTFIL data sets 25 records at a time. C1 will have records 1-25. C2 will have records 26-50. C3 will have records 51-77. C1 will have 25 records. C2 will have 25 records. C3 will have 27 records. C1, C2 and C3 will have contiguous records.

Suppressing Page Ejects (BLKCCH1, BLKCCH2, BLKCCT1)

Introduction

BLKCCH1, BLKCCH2 and BLKCCT1 are new operands of the OUTFIL statement that allow you to suppress selected page ejects in your OUTFIL reports.

- BLKCCH1 allows you to avoid forcing a page eject at the start of the report header; the ANSI carriage control character of '1' (page eject) in the first line of the report header (HEADER1) is replaced with a blank.
- BLKCCH2 allows you to avoid forcing a page eject at the start of the first page header; the ANSI carriage control character of '1' (page eject) in the first line of the first page header (HEADER2) is replaced with a blank.
- BLKCCT1 allows you to avoid forcing a page eject at the start of the report trailer; the ANSI carriage control character of '1' (page eject) in the first line of the report trailer (TRAILER1) is replaced with a blank.

Detailed Description and Syntax

You can use the BLKCCH1, BLKCCH2 and BLKCCT1 operands in the OUTFIL statement.

BLKCCH1

```
>>-BLKCCH1----><
```

Specifies that the ANSI carriage control character of '1' (page eject) in the first line of the report header (HEADER1) is to be replaced with a blank. You can use BLKCCH1 to avoid forcing a page eject at the start of the report header.

If BLKCCH1 is specified without HEADER1, it will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=RPT1,BLKCCH1,
    HEADER1=(30:'January Report',4/)
```

Default for BLKCCH1: None; must be specified.

BLKCCH2

```
>>-BLKCCH2---><
```

Specifies that the ANSI carriage control character of '1' (page eject) in the first line of the first page header (HEADER2) is to replaced with a blank. You can use BLKCCH2 to avoid forcing a page eject at the start of the first page header. BLKCCH2 does not prevent a page eject for the second and subsequent page headers.

If BLKCCH2 is specified without HEADER2, it will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=RPT2,
* Do page eject for HEADER1, but not for first HEADER2.
    HEADER1=(30:'January Report',2/),
    BLKCCH2,HEADER2=(5:'Account Number',25:'Name')
```

Default for BLKCCH2: None; must be specified.

BLKCCT1

```
>>-BLKCCT1---><
```

Specifies that the ANSI carriage control character of '1' (page eject) in the first line of the report trailer (TRAILER1) is to be replaced with a blank. You can use BLKCCT1 to avoid forcing a page eject at the start of the report trailer.

If BLKCCT1 is specified without TRAILER1, it will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=RPT3,BLKCCT1,
    TRAILER1=(5:'Grand Total is ',TOT=(21,10,ZD,M5))
```

Default for BLKCCT1: None; must be specified.

Example 1

```
  OUTFIL HEADER1=('Account Report',5X,DATE=(4MD/),/,X,/),
         HEADER2=('Page ',PAGE=(EDIT=(IT)),/,X,/,
           'Account',25:'Locations',/),
         BLKCCH2,
         BUILD=(1,8,25:13,3,ZD,M10,LENGTH=9),
         TRAILER1=(/,'Total Locations',
           25:TOT=(13,3,ZD,M10,LENGTH=9)),
         BLKCCT1
```

This example shows how BLKCCH2 and BLKCCT1 can be used to selectively suppress page ejects in a report.

A two-page version of the output report might look like this with the ANSI carriage control characters shown:

```
1Account Report     2006/03/21


 Page  1

 Account                 Locations

 10023178                        5
 18003219                        2
 23588320                       10
 ...
1Page  2

 Account                 Locations

 47890031                        3
 52831059                       11
 58729354                        4

 Total Locations               643
```

We allow the page eject ('1') for HEADER1 by not specifying BLKCCH1. We use BLKCCH2 to suppress the page eject for the first page of HEADER2, but not for the second (and subsequent) pages of HEADER2. We use BLKCCT1 to suppress the page eject for TRAILER1. Thus the report header (HEADER1) and the first page header (HEADER2) appear on the same page (page 1) instead of on separate pages, and the second page header (HEADER2) and the report trailer (TRAILER1) appear on the same page (page 2) instead of on separate pages.

Without BLKCCH2, the first page header would appear as:

```
1Page  1
```

resulting in a page eject after the three report header lines.

Without BLKCCT1, the report trailer would appear as:

```
1
 Total Locations               653
```

resulting in a page eject before the two report trailer lines.

Symbols for Columns (sym:)

Introduction

A symbol can now be used for an output column in the INREC, OUTREC and OUTFIL statements. For example, if Start_address,18 is defined in SYMNAMES, Start_address: can be used for 18:.

Detailed Description

You can use symbol: for an output column wherever you can use c: for an output column in the BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY, HEADERx and TRAILERx operands of the INREC, OUTREC and OUTFIL statements.

A symbol for p or p,m or p,m,f results in substitution of p: for symbol: (output column).

As an example of how symbols can be used for output columns, if you specify the following SYMNAMES statements:

```
Acct,27,10
Dept,38,15
```

This DFSORT control statement:

```
  INREC OVERLAY=(Acct:Acct,JFY=(SHIFT=LEFT),
    Dept:Dept,JFY=(SHIFT=LEFT))
```

will be transformed to:

```
 INREC OVERLAY=(27:27,10,JFY=(SHIFT=LEFT),38:38,15,JFY=(SHIFT=LEFT))
```

Example 1

```
...
//SYMNAMES DD *
Col1,5
Col2,25
Col3,45
End_rec,80
First,28,10
Last,7,15
Amount,45,4,FS
/*
//SYSIN DD *
  OPTION COPY
  OUTFIL REMOVECC,
    HEADER2=(Col1:'First',Col2:'Last',Col3:'Amount',/,
             Col1:10'-',Col2:15'-',Col3:6'-'),
    BUILD=(Col1:First,Col2:Last,
           Col3:Amount,EDIT=(IIT.TT),End_rec:X),
    TRAILER2=(Col1:5'=',Col3:6'=',/,
              Col1:'Total',Col3:TOT=(Amount,EDIT=(IIT.TT)))
/*
```

This example illustrates how you can use symbols for output columns in your OUTFIL report operands. The Col1, Col2, Col3 and End_rec symbols are used for output columns (sym:) in various places.

The report might look like this:

```
    First               Last               Amount
    ----------          ---------------    ------
    Buffy               Summers             12.34
    Xander              Harris               5.21
    Willow              Rosenberg           58.39
    Cordy               Chase                0.15
    Rupert              Giles               37.48
    =====                                   ======
    Total                                   113.57
```

System symbol string constants (S'string')

Introduction

A symbol can now be used for a new system symbol string constant. symbol,S'string' can be used to define a string containing any combination of EBCDIC characters and system symbols you want to use to form a character string. For example, if whererun,S'&JOBNAME. on &SYSPLEX' is defined in SYMNAMES, whererun can be used for the resulting constant. You can use dynamic system symbols such as &JOBNAME, &DAY, and so on, system-defined static system symbols such as &SYSNAME, &SYSPLEX, and so on, and installation-defined static system symbols specified by your installation in an IEASYMxx member of SYS1.PARMLIB.

Detailed Description

A symbol for a system symbol string can be used in DFSORT and ICETOOL control statements where a symbol for a character string (C'string') can be used. You can specify a symbol for a system symbol string in SYMNAMES using:

```
symbol,S'original_string'
```

or

```
symbol,s'original_string'
```

The original_string can contain any combination of EBCDIC characters and system symbols (&SYMBOL or &SYMBOL.) you want to use to form a character string. DFSORT will replace each system symbol in the system symbol string with its substitution text to create a character string in the format C'result_string'.

For example, you could specify the following symbol statement in SYMNAMES:

```
Rpt_hdr,S'  Job &JOBNAME. was run on Sysplex &SYSPLEX. on &LWDAY'
```

If you used this symbol statement in a job named TEST2 that you ran on sysplex MAS3 on a Thursday, DFSORT would transform it into the following symbol statement:

```
Rpt_hdr,C'  Job TEST2 was run on Sysplex MAS3 on THU'
```

&JOBNAME. was replaced with 'TEST2', &SYSPLEX. was replaced with 'MAS3' and &LWDAY was replaced with 'THU'.

If you used this same symbol statement in a job named BIGTEST that you ran on sysplex MAS2 on a Monday, DFSORT would transform it into the following symbol statement:

```
Rpt_hdr,C'  Job BIGTEST was run on Sysplex MAS2 on MON'
```

This time &JOBNAME. was replaced with 'BIGTEST', &SYSPLEX. was replaced with 'MAS2' and &LWDAY was replaced with 'MON'.

You could use the Rpt_hdr symbol in an OUTFIL statement like this:

```
  OUTFIL HEADER2=(Rpt_hdr,5X,'Page ',PAGE=(EDIT=(TTT)),/)
```

and get the heading you needed based on the setting of the system symbol values where and when the job was run.

The types of system symbols you can use in a system symbol string are:

- Dynamic system symbols like &YYMMDD, &LYYMMDD, &HHMMSS, &LHHMMSS, &DAY, &LDAY, &HR, &LHR, &JDAY, &LJDAY, &JOBNAME, &MIN, &LMIN, &MON, &LMON, &SEC, &LSEC, &WDAY, &LWDAY, &YR2, &LYR2, &YR4 and &LYR4.

- System-defined static system symbols like &SYSCLONE, &SYSNAME, &SYSPLEX, &SYSR1 and &SYSALVL.
- Installation-defined static system symbols specified by your installation in an IEASYMxx member of SYS1.PARMLIB.

Note: System symbols must be specified using all uppercase characters. Lowercase characters are not recognized as system symbols. For example, &JOBNAME is recognized as a system symbol, but &jobname and &Jobname are not.

You can use all three types of system symbols separately or in combination within a system symbol string. You can also use all of the system symbol string building facilities such as substring notation, concatenation, embedded blanks, and so on, in a system symbol string. Thus, system symbol strings can be very simple like this one:

```
Sysname,s'&SYSNAME'
```

Or more complex like this one:

```
Dsn1,s'ACCT.&SYSNAME(3:2)..D&LJDAY'
```

For more information on system symbols, see *z/OS MVS Initialization and Tuning Reference.*

Note: JCL symbols and IPCS symbols are not system symbols and will not be recognized or replaced in a system symbol string.

s'original_string' will be treated like S'original_string'.

If you want to include a single apostrophe in the system symbol string, you must specify it as two single apostrophes.

DFSORT uses the ASASYMBM service to transform S'original_string' into C'result_string'. If your system symbol string has errors involving symbol names, substring notation, and so on, ASASYMBM transforms the symbol system string to a character string according to its rules for substitution. For example, if you specify these symbol statements:

```
Ok,S'&YR4(1:2).&SYSNAME.'
Bad,S'&YR4(1.2).&SYSNAM.'
```

where the valid substring notation 1:2 and the known system symbol &SYSNAME. are used for Ok, but the invalid substring notation 1.2 and the unknown system symbol &SYSNAM. are used for Bad, the resulting symbol statements are:

```
Ok,C'20EDS3'
Bad,C'2006(1.2).&SYSNAM.'
```

See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for complete details on the ASASYMBM service.

After a symbol statement containing a system symbol string (S'original_string') is transformed into a symbol statement containing the resulting character string (C'result_string'), it will be error checked and processed like any other symbol statement containing a character string. See the description of character string above for details.

If a SYMNOUT data set is specified, it will show the symbol statement containing the original system symbol string as well as the transformed symbol statement containing the resulting character string. This can be helpful for debugging "errors" or unexpected results in your system symbol strings.

Example 1

```
...
//SYMNAMES DD *
Title,S'Jobname: &JOBNAME., Sysplex: &SYSPLEX., System: &SYSNAME.,'
Totday,S'Total for &WDAY.:'
/*
//SYSIN    DD *
  OPTION COPY
  OUTFIL REMOVECC,
   HEADER2=(Title,X,' Page: ',PAGE=(EDIT=(TT)),/),
   TRAILER1=(/,Totday,20:TOT=(21,4,FS,TO=FS,LENGTH=5))
/*
```

This example illustrates how you can use system symbol strings in your OUTFIL report operands. The Title and Totday constants incorporate the values for system symbols &JOBNAME, &SYSPLEX, &SYSNAME and &WDAY. If job TESTRUN ran on system EDS3 within sysplex MAS3 on Thursday, the report might look like this:

```
Jobname: TESTRUN, Sysplex: MAS3, System: EDS3,  Page: 01

Denver               -56
Boulder             +123
San Jose              +8

Total for THU:        75
```

INREC with SELECT and SPLICE

Introduction

SELECT and SPLICE now allow you to use an INREC statement to reformat your records before they are selected or spliced. All of the operands of the INREC statement (PARSE, BUILD, OVERLAY, IFTHEN and IFOUTLEN) are now available with SELECT and SPLICE.

Detailed Description

You can use INCLUDE, OMIT, INREC, OPTION, SORT and OUTFIL statements with **SELECT** providing you observe these rules:

- You can use an INCLUDE or OMIT statement to remove input records **before** SELECT processing.
- You can use an INREC statement to reformat input records **before** SELECT processing. You can use INREC's PARSE, BUILD, OVERLAY, IFTHEN or IFOUTLEN functions. If your INREC statement changes the starting position of an ON field, you must specify the new starting position for that ON field. For example, if your input records have a CH key at positions 1-5 and you use an INREC statement like this:

  ```
  INREC FIELDS=(25:1,50)
  ```

  you must specify ON(25,5,CH) instead of ON(1,5,CH).
- If you specify a SORT statement, you must specify each ON field as a p,m,f,A sort field and these sort fields must be in the same order as the ON fields. After these sort fields, you can specify additional p,m,f,A or p,m,f,D sort fields. The additional sort fields will be used for sorting, but not for selecting. For example, if you use a SELECT statement like this:

  ```
  SELECT FROM(IN) TO(OUT) ON(21,5,CH) FIRST USING(CTL1)
  ```

  you can use a SORT statement in CTL1CNTL like this:

```
SORT FIELDS=(21,5,CH,A,41,6,CH,D)
```

The records will be sorted by the 21,5,CH,A field and the 41,6,CH,D field, but only selected by the 21,5,CH field. This would allow you to select the highest 41,6,CH value for each 21,5,CH value.

- If you specify TO(outdd) without DISCARD(savedd), you can further process the outdd records **after** SELECT 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** SELECT 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** SELECT processing using two (and only two) 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,INCLUDE=(21,3,ZD,GT,+25)
  ```

You can use INCLUDE, OMIT, INREC, OPTION and OUTFIL statements with **SPLICE** providing you observe these rules:

- You can use an INCLUDE or OMIT statement to remove input records **before** SPLICE processing.
- You can use an INREC statement to reformat input records **before** SPLICE processing; the base and overlay records are reformatted according to the INREC statement. You can use INREC's PARSE, BUILD, OVERLAY, IFTHEN or IFOUTLEN functions. If your INREC statement changes the starting position of an ON field or WITH field, you must specify the new starting position for that ON field or WITH field. For example, if your input records have a CH key at positions 1-5 and a WITH field at positions 6-8, and you use an INREC statement like this:

  ```
  INREC FIELDS=(31:1,50)
  ```

  you must specify ON(31,5,CH) instead of ON(1,5,CH), and WITH(36,3) instead of WITH(6,3).

- You can further process the outdd records associated with TO(outdd) **after** SPLICE 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 they have been spliced, with a statement like this:

  ```
  OUTFIL FNAMES=OUT1,FTOV,VLTRIM=X'40'
  ```

Example 1

```
SELECT FROM(INB) TO(OUTB) ON(8,10,CH) -
  FIRSTDUP USING(BIRD)
```

This example shows how you can use USING(xxxx) to supply an OMIT statement to remove certain input records, and an INREC statement to reformat certain input records, before SELECT processing begins.

Let's say the INB data set looks like this:

```
TYPE1  CROWS
TYPE1  PIGEONS
TYPE1  HAWKS
TYPE1  ROBINS
TYPE1  SPARROWS
TYPE1  CHICKENS
TYPE1  RAVENS
TYPE2            PIGEONS
TYPE2            ROBINS
TYPE2            HAWKS
TYPE2            TERNS
TYPE3  EAGLES
TYPE3  STARLINGS
```

We want to remove the TYPE3 records and then select one record for each type of bird that appears in both a TYPE1 and a TYPE2 record. We could use the following control statements in BIRDCNTL:

```
* Remove the TYPE3 records.
   OMIT COND=(1,5,CH,EQ,C'TYPE3')
* Reformat the TYPE2 records to place the bird name in the
* same place as in the TYPE1 records so we can match them.
   INREC IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE2'),
      BUILD=(8:18,10))
```

The OUTB data set would look like this:

```
TYPE1  HAWKS
TYPE1  PIGEONS
TYPE1  ROBINS
```

Example 2

This example shows how you can use USING(xxxx) to supply a SORT statement to alter the records that are selected.

```
SELECT FROM(IN) TO(OUT) ON(1,5,CH) FIRST USING(CTL1)
```

Let's say the IN data set looks like this:

```
FRANK 00015
FRANK 00012
FRANK 00018
FRANK 00005
VICKY 00022
VICKY 00028
VICKY 00002
VICKY 00015
```

We want to select the record with each name that has the highest count. If we just used ON(1,5,CH) without any CTL1CNTL statements, we'd get the first record without regard to the count. The OUT data set would look like this:

```
FRANK 00015
VICKY 00022
```

To get the record with the highest count, we can use the following SORT statement in CTL1CNTL:

```
  SORT FIELDS=(1,5,CH,A,7,5,ZD,D)
```

The records will be sorted in ascending order on the name field, and in descending order on the count field. By sorting descending on the count, we ensure that the record with the highest count is the first record for each name. Thus, when ON(1,5,CH) selects the first record, it will be the one with the highest count. The OUT data set will look like this:

```
FRANK 00018
VICKY 00028
```

Example 3

```
...
//TOOLIN DD *
SPLICE FROM(MAST) TO(OUT) WITH(1,7) -
  WITH(13,4) ON(20,3,CH) -
  WITH(23,3) WITH(26,3) WITHALL USING(CTL1)
/*
//CTL1CNTL DD *
* Before SPLICE:
* Set up fields in base (A) records.
* Add 'B' id in position 33.
   INREC IFTHEN=(WHEN=(1,1,CH,EQ,C'A'),
           BUILD=(8:14,5,17:31,3,
             20:11,3,29:34,4,33:C'B')),
* Set up fields in overlay (H) records.
* Add 'V' id in position 33.
         IFTHEN=(WHEN=(1,1,CH,EQ,C'H'),
           BUILD=(1:7,7,13:18,4,
             20:4,3,23:1,3,26:22,3,33:C'V'))
/*
```

This example shows how you can use the WITHALL operand to tell ICETOOL to splice data together for a single record of one type (A records) and multiple records of another type (H records), in the same input data set, that all have the same ON field (duplicate records). It also shows how to ensure that duplicates of the second type without a match of the first type are not written to the output data set. IFTHEN clauses are used in an INREC statement to reformat the two types of records appropriately before they are sorted and spliced.

If the MAST data set looks like this:

```
A0000B0000KRSC0000D000000E0000F00G000
A1111B1111FLYC1111D111111E1111F11G111
H02KRSI000002J002K002L02
H03FLYI000003J003K003L03
H04VQXI000004J004K004L04
H05FLYI000005J005K005L05
H06KHNI000006J006K006L06
H07KRSI000007J007K007L07
H08FLYI000008J008K008L08
H09KHNI000009J009K009L09
```

The OUT data set will look like this:

```
I000003C1111K003F11FLYH03L03G111
I000005C1111K005F11FLYH05L05G111
I000008C1111K008F11FLYH08L08G111
I000002C0000K002F00KRSH02L02G000
I000007C0000K007F00KRSH07L07G000
```

Larger PD and ZD Sort and Merge Fields

Introduction

The maximum length for a PD or ZD sort or merge field has been raised to 256.

Detailed Description

The maximum length for signed packed decimal (PD) and signed zoned decimal (ZD) fields used in the SORT and MERGE statements has been raised from 32 bytes to 256 bytes. Your PD and ZD sort and merge fields can now be 1-256 bytes.

Example 1

```
SORT FIELDS=(21,200,ZD,A,
      3001,256,PD,D)
```

This example illustrates how you can use larger ZD and PD fields in the SORT statement. It sorts ascending on a 200-byte ZD field starting in position 21 and descending on a 256-byte PD field starting in position 3001.

Higher Ending Position for Sum Fields

Introduction

The maximum position for the end of a sum field has been raised to 32752.

Detailed Description

The maximum end position for a sum field has been raised from 4092 to 32752. A sum field can extend up to, but not beyond, byte 32752.

Example 1

```
SUM FIELDS=(15021,8,PD,15011,4,FI)
```

This example illustrates how you can use higher ending positions for sum fields. It sums an 8-byte PD field in positions 15021-15028 and a 4-byte FI field in positions 15011-15014.

DSA Run-Time Option

Introduction

DSA can now be specified as a run-time option. This allows you to adjust the maximum amount of storage available to DFSORT for dynamic storage adjustment of individual Blockset sort applications when SIZE/MAINSIZE=MAX is in effect.

Detailed Description and Syntax

You can now specify DSA=n as an EXEC PARM, DFSPARM PARM and OPTION option as well as an installation option. If the DSA=n run-time option is specified for a particular sort application, it overrides the DSA=n installation option for that application.

```
>>-DSA=n------------><
```

Temporarily overrides the DSA installation option, which specifies the maximum amount of storage available to DFSORT for dynamic storage adjustment of a Blockset sort application when SIZE/MAINSIZE=MAX is in effect. If you specify a DSA value greater than the TMAXLIM value in effect, you allow DFSORT to use more storage than the TMAXLIM value if doing so should improve performance. The amount of storage DFSORT uses is subject to the DSA value as well as system limits such as region size. However, whereas DFSORT always tries to obtain as much storage as it can up to the TMAXLIM value, DFSORT only tries to obtain as much storage as needed to improve performance up to the DSA value.

The performance improvement from dynamic storage adjustment usually provides a good tradeoff against the increased storage used by DFSORT. On storage constrained systems, however, the DSA value should be set low enough to prevent unacceptable paging.

n specifies that DFSORT can dynamically adjust storage to improve performance, subject to a limit of n MB. n must be a value between 0 and 2000. If n is less than or equal to the TMAXLIM value in effect, n is set to 0 to indicate that storage will not be dynamically adjusted.

Default: Usually the installation default. See Appendix B "Specification/Override of DFSORT Options" in *z/OS DFSORT Application Programming Guide* for full override details.

Example 1

```
OPTION DSA=100
```

This example illustrates how you can specify a DSA value at run-time to override the DSA installation value for a particular sort application. The OPTION statement specifies that DFSORT can dynamically adjust storage to improve performance for this application, subject to a limit of 100 megabytes.

F and C Signs for PD Output Fields (PDF, PDC)

Introduction

The BUILD, OVERLAY, IFTHEN BUILD, and IFTHEN OVERLAY operands of the INREC, OUTREC and OUTFIL statements and the TRAILERx and HEADERx operands of the OUTFIL statement now allow you to use new TO=PDF and TO=PDC options to convert numeric values to PD values with F or C for the positive sign, respectively. The TO=PDC option is equivalent to the existing TO=PD option.

Detailed Description

TO=PDF and TO=PDC can be used in the BUILD, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY, TRAILERx and HEADERx operands to specify an output format in the same way TO=PD can be used in those operands. For PD or PDC output, C is used as the positive sign (e.g. P'+12' = X'012C') and D is used as the negative sign (e.g. P'-12' = X'012D'). For PDF output, F is used as the positive sign (e.g. P'+12' = X'012F' and D is used as the negative sign (e.g. P'-12' = X'012D').

Example 1

```
OPTION COPY
OUTREC BUILD=(5,3,ZD,ADD,+20,TO=PDF,LENGTH=4,X,
    18,2,BI,TO=PDF,80:X)
OUTFIL REMOVECC,
  TRAILER1=(5:COUNT=(TO=PDF,LENGTH=6),
     20:TOT=(1,4,PD,TO=PDF,LENGTH=4))
```

This example illustrates how you can use TO=PDF to get an F sign for positive PD output values instead of a C sign. The OUTREC statement uses TO=PDF for arexp,to and p,m,f,to. The OUTFIL statement uses TO=PDF for COUNT=(to) and TOT=(p,m,f,to). Note that TO=PD or TO=PDC would have given a C sign for positive PD output values, whereas TO=PDF gives an F sign.

Spaces Between Title Elements (TBETWEEN)

Introduction

DISPLAY and OCCUR now allow you to use a new TBETWEEN(n) operand to specify the number of blanks between title elements (title, page number, date, time).

Detailed Description and Syntax

By default, 8 blanks appear between title elements in ICETOOL DISPLAY and OCCUR reports. You can now use the new TBETWEEN(n) operand to change the number of blanks between title elements in these reports.

```
>>--TBETWEEN(n)-------><
```

Specifies the number of blanks to be used between title elements (overriding the default of 8). TBETWEEN(n) can appear between any of the following title elements:

- title: TITLE('string')
- page number: PAGE
- date: DATE, DATE(abcd), DATENS(abc), YDDD(abc), YDDDNS(ab)
- time: TIME, TIME(abc), TIMENS(ab)

For example, if TBETWEEN(n) is not specified, 8 blanks appear between the title and date and between the date and page number, whereas if TBETWEEN(4) is specified, 4 blanks appear between the title and date and between the date and page number.

n is the number of blanks between the title elements. n can be 0 to 50.

Example 1

```
OCCUR FROM(IN) LIST(RPT) BLANK TBETWEEN(3) -
  TITLE('Tools on hand') PAGE DATENS(MD4) TIME(12:) -
  HEADER('Type of Tool') ON(11,10,CH) -
  HEADER('On hand') ON(VALCNT,U07)
```

This example shows how you can use TBETWEEN(3) to change the number of blanks between title elements from 8 to 3. The RPT output might look like this:

```
Tools on hand   - 1 -   03242006   03:15:07 pm

Type of Tool    On hand
------------   --------
Chisel                4
Hammer                6
Saw                  13
Wrench               11
```

Zero Pointers for RECORD in 24-Bit Parmlist

Introduction

DFSORT now accepts and ignores zero values in the starting and ending address of the RECORD statement image in the 24-Bit Parameter List. You can set these addresses to zero if you don't want to pass a control statement to DFSORT using the third and fourth words of the parameter list .

Note: We recommend using the more flexible Extended Parameter List rather than the 24-Bit Parameter List.

Detailed Description and Syntax

If you don't want to pass a control statement using the third and fourth words of the 24-Bit Parameter List (generally used for the RECORD statement), you can set those addresses to zero. You must pass a starting and ending address of a control statement (e.g. SORT) in the first and second words of the 24-Bit Parameter List, but you can choose not to pass a starting and ending address of a control statement in the third and fourth words by setting those words to zero.

The format of the beginning of the 24-Bit Parameter List now looks as follows:

```
Word1 |X'00'|Start address of statement               |
Word2 |X'00'|End address of statement                 |
Word3 |X'00'|Start address of statement (zero if none)|
Word4 |X'00'|End address of statement   (zero if none)|
Word5 |X'00'|Address of E15 or E32      (zero if none)|
...
```

Word1: The starting address of a statement image. The first byte must contain X'00'. The last three bytes must contain a valid address for the start of a statement.

Word2: The ending address of a statement image. The first byte must contain X'00'. The last three bytes must contain a valid address for the end of the statement which starts at the address in Word1.

Word3: The starting address of a statement image, if any; otherwise, all zeros. The first byte must contain X'00'. The last three bytes must contain a valid address for the start of a statement, or all zeros.

Word4: The ending address of a statement image, if any; otherwise, all zeros. The first byte must contain X'00'. The last three bytes must contain a valid address for the end of the statement which starts at the address in Word3, or all zeros if Word3 is all zeros.

Word5: The address of the E15 or E32 routine that your program has placed in storage, if any; otherwise all zeros. The first byte must contain X'00'. The last three bytes must contain a valid address for the E15 or E35 routine, or all zeros.

Example 1

```
...
        LA   1,PARLST               LOAD ADDRESS IN R1
        LINK EP=SORT                INVOKE DFSORT
...
PARLST  DC   X'80',AL3(ADLST)       ADDRESS OF LIST
...
        CNOP 2,4                    ALIGN TO BOUNDARY
ADLST   DC   AL2(LISTEND-LISTBEG)   LIST LENGTH
LISTBEG DC   A(SORTA)               START OF SORT STMT
        DC   A(SORTZ)               END OF SORT STMT
        DC   A(0)                   NO IMAGE
        DC   A(0)                   NO IMAGE
        DC   A(0)                   NO E15 ROUTINE
        DC   A(0)                   NO E35 ROUTINE
LISTEND EQU  *
SORTA   DC   C' SORT FIELDS=(10,15,CH,A)'
SORTZ   DC   C' '
...
```

This example show how you can use zero addresses for the third and fourth words of the 24-Bit Parameter List if you don't want to pass a statement to DFSORT with those words.

New and Changed Messages

This section shows messages that have been added, or changed significantly, for PTFs UK90007 and UK90006. Refer to *z/OS DFSORT Messages, Codes and Diagnosis Guide* for general information on DFSORT messages.

ICE007A

ICE007A Syntax Error

Explanation: Critical. A control statement contained an error in syntax.

System Action: The program terminates.

Programmer Response: Check the control statements for syntax errors. Some of the more common syntax errors are:

- Unbalanced parenthesis
- Missing comma
- Embedded blank
- Invalid format type
- Invalid operator
- Invalid constant
- Continuation of DFSPARM PARM options using the continuation column.
- Symbol used where it is not allowed
- Parsed field (%nn) used where it is not allowed

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:

- A PARSE, FIELDS, BUILD, or OVERLAY operand was specified twice. Example:

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

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

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

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

```
INREC OVERLAY=(21:C'A'),IFOUTLEN=50
```

- 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 or OVERLAY. Example:

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

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

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

- 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=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.

ICE109A

ICE109A SUM FIELD DISPLACEMENT OR LENGTH VALUE ERROR

Explanation: Critical. A sum field definition on a SUM control statement contained an invalid length or displacement (position) value.

System Action: The program terminates.

Programmer Response: Make sure that the length and position values in the FIELDS operand of the SUM control statement were specified correctly. For BI and FI, length must be 2, 4, or 8 bytes; for PD, length must be 1 through 16 bytes; for ZD, length must be 1 through 31 bytes; for FL, length must be 4, 8 or 16 bytes. Make sure that the position plus length of each field does not exceed 32753.

ICE111A

ICE111A REFORMATTING FIELD ERROR

Explanation: Critical. The FIELDS, BUILD, OVERLAY, IFTHEN BUILD or IFTHEN OVERLAY operand of an INREC or OUTREC statement, or the OUTREC, BUILD, OVERLAY, IFTHEN BUILD or IFTHEN OVERLAY operand of an OUTFIL statement, contained an invalid column, separator, position, length, keyword, pattern, sign, constant or value. Some common errors are:

- A 0 value was used.
- A null value was used where it was not permitted.
- A null string, pattern, or sign was used.
- A column was greater than 32752, or was followed by another column.
- A position plus length was greater than 32753.
- DATE=(abcd) or DATENS(abc) was specified with a, b or c not M, D, Y or 4, with M, D, Y or 4 specified more than once, or with Y and 4 both specified.

- YDDD=(abc) or YDDDNS=(ab) was specified with a or b not D, Y or 4, with D, Y or 4 specified more than once, or with Y and 4 both specified.
- TIME=(abc) or TIMENS=(ab) was specified with ab not 12 or 24.
- The length (m for p,m or FIXLEN=m for %nn) for a hexadecimal field was greater than 16376.
- A repetition factor was greater than 4095 for a separator, or a character or hex constant was longer than 256 bytes.
- An invalid digit or an odd number of digits was specified for a hexadecimal constant.
- The length (m for p,m or FIXLEN=m for %nn) for a Y2 format field was not 2 for Y2C, Y2Z, Y2P, Y2S or Y2PP, or 1 for Y2D, Y2B or Y2DP, or 3-6 for Y2T, Y2W, Y2TP or Y2WP, or 2-3 for Y2U, Y2X, Y2UP or Y2XP, or 3-4 for Y2V, Y2Y, Y2VP or Y2YP.
- The length (m for p,m or FIXLEN=m for %nn) for an edit field was less than 2 or greater than 8 for PD0.
- The length (m for p,m or FIXLEN=m for %nn) for an edit field was greater than 8 for BI or FI, 16 for PD, 31 for ZD, 32 for CSF/FS or 44 for UFF or SFF.
- The length (m for p,m or FIXLEN=m for %nn) for an edit field was not 4 for DT1, DT2, DT3, TM1, TM2, TM3, or TM4. The length (m for p,m or FIXLEN=m for %nn) for an edit field was not 8 for DC1, DC2, DC3, TC1, TC2, TC3, TC4. DE1, DE2, DE3, TE1, TE2, TE3, or TE4.
- The length (m for p,m or FIXLEN=m for %nn) for an edit field was not 4 or 8 for FL.
- %nn,H or %nn,F or %nn,D was specified.
- More than 31 digits or 44 characters were specified in an edit pattern.
- SIGNz (where z is not S) was specified with Mn or without EDIT or EDxy.
- x, y, or z in EDxy or SIGNz were the same character.
- The value for LENGTH was greater than 44.
- NOMATCH was specified after p,m or %nn rather than after CHANGE.
- The length for a lookup input field was greater than 64 with character or hexadecimal find constants, or greater than 1 with find bit constants.
- The length for a lookup output field was greater than 64.
- The length for a find constant was greater than the lookup input field length.
- A find constant was not a character, hexadecimal, or bit constant.
- The length for a set constant or set field (m for p,m or FIXLEN=m for %nn) was greater than the lookup output field length.
- An invalid character was specified in a find bit constant, or the number of bits for a find bit constant was not 8.
- A set constant was not a character or hexadecimal constant.
- The length for a sequence number was greater than 16.
- The value for START was greater than 100000000000.
- The value for INCR was greater than 10000000.
- The length (m for p,m or FIXLEN=m for %nn) for a RESTART field was greater than 256 bytes.
- A position without length (p without m) was specified for OVERLAY or IFTHEN OVERLAY.
- A / was specified for OVERLAY or IFTHEN OVERLAY.

System Action: The program terminates.

Programmer Response: Correct the invalid value.

ICE114A

ICE114A INVALID COMPARISON

Explanation: Critical. One of the following situations exists:

- The COND operand of an INCLUDE or OMIT statement, or the INCLUDE or OMIT operand of an OUTFIL statement, or the IFTHEN WHEN operand of an INREC, OUTREC, or OUTFIL statement, contained an invalid field-to-field, field-to-mask, or field-to-constant comparison, or used a field with NUM that was not FS, CSF, PD or ZD.
- FORMAT=SS was specified after COND in an INCLUDE or OMIT statement.
- Locale processing was required, but a character (CH) field to binary (BI) field comparison was specified. Locale processing does not support the comparison of a CH to BI field.

System Action: The program terminates.

Programmer Response: Make sure that all of the comparisons are valid. If you specified FORMAT=SS after COND in an INCLUDE or OMIT statement, respecify it before COND. DFSORT's locale processing might eliminate the need for CH to BI comparisons. See *z/OS DFSORT Application Programming Guide* for information relating to locale processing. If a CH to BI comparison is needed, specify run time option LOCALE=NONE.

ICE185A

ICE185A AN xnnnn ABEND WAS ISSUED BY DFSORT, ANOTHER PROGRAM OR AN EXIT (PHASE m y)

Explanation: Critical. DFSORT detected a system or user abend while it's ESTAE routine was in effect. The abend may have been issued by DFSORT, another program that called DFSORT, or a user exit called by DFSORT. Although DFSORT detected the abend, DFSORT is not necessarily the cause of the abend.

This message gives details about the abend as follows:

- x is the abend type, either S for system or U for user
- nnnn is the abend code

This message gives details about DFSORT's application mode (if known at the time of the error) and phase at the time DFSORT detected the abend:

- m is S, M, or C for a Blockset sort, merge, or copy, respectively. m is P when the Peerage/Vale technique was used. m can also be blank if the abend occurred before DFSORT determined the application type.
- y is the phase number 0, 1, 2, 3, or 4.

System Action: The program terminates.

Programmer Response: Use the information in the abend dump to determine if DFSORT, a calling program or a user exit caused the abend, and take appropriate action.

For system abends, see the appropriate systems codes document.

User abends are program specific. If Unnnn is reported in the message, nnnn in the range 1000-1675 or 2222 may or may not have been issued by DFSORT. nnnn outside that range was not issued by DFSORT.

ICE189A

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

Explanation: Critical. Blockset was required for one of the following:

- LOCALE processing
- OUTFIL processing
- Y2x, Y2xx, PD0, FS, CSF, UFF, or SFF format
- PARSE, OVERLAY, IFTHEN, or IFOUTLEN processing
- INREC or OUTREC processing with one of the following:
  - p,m,HEX
  - p,HEX
  - p,m,TRAN=LTOU
  - p,TRAN=LTOU
  - p,m,TRAN =UTOL
  - p,TRAN=UTOL
  - p,m,TRAN=ALTSEQ
  - p,TRAN=ALTSEQ
  - p,m,f
  - p,m,lookup
  - p,m,JFY=(...)
  - p,m,SQZ=(...)
  - %nn or %n
  - SEQNUM
  - DATE1, DATE1(c), DATE1P, DATE2, DATE2(c), DATE2P, DATE3, DATE3(c), DATE3P, or +n and -n variations (for example, DATE1-20 or DATE3P+20).
  - DATE4, DATE, DATE=(abcd), DATENS=(abc), YDDD=(abc) or YDDDNS=(ab),
  - TIME1, TIME1(c), TIME1P, TIME2, TIME2(c), TIME2P, TIME3, TIME3P, TIME, TIME=(abc), or TIMENS=(ab)
  - +n
  - -n
  - (...)
- A VSAM extended addressability data set
- To set the SORTOUT LRECL from the L3 length (without E35, INREC or OUTREC), the OUTREC length or the INREC length, with SOLRF in effect
- an FL format sort field with NOSZERO in effect

- VLLONG in effect and SORTOUT present
- VSAMEMT in effect for a sort or merge with VSAM input
- The same VSAM data set was specified for both input and output
- An HFS file was specified for input or output
- A tape data set with a block size greater than 32760 bytes was specified for input or output
- SDB=LARGE or SDB=INPUT was in effect and DFSORT selected a block size greater than 32760 bytes for a tape output data set
- VLSHRT in effect with a SUM statement
- Position plus length for a control field exceeded 4093
- Position plus length for a summary field exceeded 4093
- ICETOOL called DFSORT for an operation involving SORTOUT, and NULLOUT=RC16 is in effect.
- DB2 Utilities called DFSORT.
- INCLUDE or OMIT processing with NUM
- A 33 to 256 byte PD, ZD, CSL, CST, CLO, CTO, ASL or AST format merge field
- A 256 byte PD format sort field with NOSZERO in effect

However, Blockset could not be used due to the reason indicated by reason code nn. See message ICE800I for the meaning of nn.

System Action: The program terminates.

Programmer Response: Correct the situation indicated by the reason code so Blockset can be used. Alternatively, you can remove the source of the requirement to use Blockset. However, this will result in the use of a less efficient technique.

ICE211I

ICE211I OLD OUTFIL STATEMENT PROCESSING USED

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

System Action: Processing continues, but OUTFIL data sets are not associated with this OUTFIL statement. If the Blockset technique is not selected, control statement errors could result from continuation of this OUTFIL statement.

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

ICE212A

ICE212A MATCH NOT FOUND FOR {*INREC|*OUTREC|ddname} IFTHEN n CHANGE FIELD: {POSITION p|PARSE x}

Explanation: Critical. In the FIELDS, BUILD, OVERLAY, IFTHEN BUILD or IFTHEN OVERLAY operand of an INREC or OUTREC statement, or the OUTREC, BUILD, OVERLAY, IFTHEN BUILD or IFTHEN OVERLAY operand of an OUTFIL statement, a CHANGE parameter was specified without a NOMATCH parameter and fixed field (p,m) value or parsed field (%x) value did not match any of the find constants.

POSITION p indicates that a fixed field (p,m) did not match any of the find constants. p is the starting position of the input field. PARSE x indicates that a parsed field (%x) did not match any of the find constants.

The specific cause of the error is identified as follows:

- *INREC and n=0 indicates that a change field in the FIELDS, BUILD or OVERLAY operand of the INREC statement caused the error.
- *OUTREC and n=0 indicates that a change field in the FIELDS, BUILD or OVERLAY operand of the OUTREC statement caused the error.
- ddname and n=0 indicates that a change field in the OUTREC, BUILD or 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 a change field in an IFTHEN BUILD or IFTHEN OVERLAY 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 a change field in an IFTHEN BUILD or IFTHEN OVERLAY 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 a change field in an IFTHEN BUILD or IFTHEN OVERLAY 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).

System Action: The program terminates. The program terminates when the first change field is encountered for which a match is not found.

Programmer Response: Correct the lookup table specified with the CHANGE parameter, or use the NOMATCH parameter to specify a constant, or input field or parsed field to be used as the output field if a match is not found. The use of a constant such as NOMATCH=(C'**') can be helpful in identifying all fixed field values or parsed field values for which a match is not found.

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 and IFTHEN, OUTREC and BUILD, OUTREC and OVERLAY, OUTREC and IFTHEN, BUILD and OVERLAY, BUILD and IFTHEN, or OVERLAY and IFTHEN were specified.

  Example:

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

- IFOUTLEN and PARSE, IFOUTLEN and BUILD, IFOUTLEN and OUTREC or IFOUTLEN and OVERLAY were specified.

  Example:

  ```
  OUTFIL OVERLAY=(21:C'A'),IFOUTLEN=50
  ```

- 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 or OVERLAY

  Example:

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

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

  Example:

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

- 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=NONE was followed by an IFTHEN clause with WHEN=INIT, WHEN=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.

ICE216A

ICE216A TOTAL LENGTH OF CONTROL FIELDS AND SUM FIELDS IS TOO LONG

Explanation: Critical. The total length of the SORT or MERGE control fields and SUM summary fields is too long for DFSORT to process, or the complexity of the application caused dynamic areas to exceed the storage allowed for them. Note that locale processing can significantly decrease the total length of the SORT or MERGE fields DFSORT can process.

System Action: The program terminates.

Programmer Response: Remove one or more control fields or summary fields, or make them shorter.

ICE221A

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

Explanation: Critical. An error was detected in a COND, INCLUDE, OMIT, or IFTHEN WHEN operand. The specific cause of the error is identified as follows:

- *INCLUDE indicates that the COND operand of the INCLUDE statement caused the error. n is 0.

- *OMIT indicates that the COND operand of the OMIT statement caused the error. n is 0.
- ddname and n=0 indicates that the INCLUDE or OMIT operand of an OUTFIL statement caused the error. ddname identifies the first data set in the associated OUTFIL group.
- *INREC indicates that an IFTHEN WHEN 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 indicates that an IFTHEN WHEN 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 WHEN 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).

One of the following errors was detected:

- the length for a field with a format other than SS was greater than 256
- the length for a PD field not used with NUM was 256
- the length for a PD0 field was less than 2 or greater than 8
- the length for a CSF or FS field not used with NUM was greater than 32
- the length for a UFF or SFF field was greater than 44
- the length for a CSL, CST, ASL, or AST field was 1
- the decimal constant for an FI field was greater than +9223372036854775807 or less than -9223372036854775808
- the decimal constant for a BI field was greater than 18446744073709551615 or less than +0
- the number of digits (including leading zeros) in the decimal constant for an FI or BI field was greater than 31
- the length for a Y2 field was not 2 for Y2C, Y2Z, Y2P or Y2S, or 1 for Y2D or Y2B, or 3-6 for Y2T or Y2W, or 2-3 for Y2U or Y2X, or 3-4 for Y2V or Y2Y
- a Y2 field was compared to another Y2 field with a different number of non-year digits
- a Y2 field was compared to a Y constant with a different number of non-year digits
- a Y2 field other than Y2S, Y2T or Y2W was compared to Y'LOW', Y'BLANKS' or Y'HIGH'
- a Y2 field was compared to a constant that was not a Y constant.

m indicates the number of the relational condition in which the error was found (starting at 1 for the first relational condition). For example, in

```
INCLUDE COND=(5,2,CH,EQ,8,2,CH,OR,
  11,257,BI,EQ,301,257,BI)
```

the second relational condition (after the OR) has the error described in the first bullet above, so m is 2.

System Action: The program terminates.

Programmer Response: Correct the field length or constant in error in relational condition m.

ICE223A

ICE223A REPORT FIELD ERROR

Explanation: Critical. The LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, or SECTIONS parameter of an OUTFIL statement contained an invalid column, report element, position, length, format, keyword, pattern, sign, or constant. Some common errors are:

- A 0 value was used.
- A null value was used where it was not permitted.
- A null string, pattern, or sign was used.
- A column was greater than 32752, preceded / or n/ (new line), or was followed by another column.
- A column overlapped the previous output field in the report record (a missing new line (/ or n/) to end the current report record and start the next one can cause this error).
- A position plus length was greater than 32753.
- DATE=(abcd) or DATENS(abc) was specified with a, b or c not M, D, Y or 4, with M, D, Y or 4 specified more than once, or with Y and 4 both specified.
- YDDD=(abc) or YDDDNS=(ab) was specified with a or b not D, Y or 4, with D, Y or 4 specified more than once, or with Y and 4 both specified.
- TIME=(abc) or TIMENS=(ab) was specified with ab not 12 or 24.
- The length for an input field or section break field was greater than 256 bytes.
- A repetition factor was greater than 4095 for a blank, character string, or hexadecimal string report element, or greater than 255 for a blank lines report element or a section skip line count.
- A character or hexadecimal constant was longer than 256 bytes.
- An invalid digit or an odd number of digits was specified for a hexadecimal string.
- The length for a statistics field was greater than 8 for BI or FI, 16 for PD, 31 for ZD, 32 for CSF/FS, or 44 for UFF or SFF.
- The length for a statistics field was not 4 or 8 for FL.
- More than 31 digits or 44 characters were specified in an edit pattern.
- SIGNz (where z is not S) was specified with Mn or without EDIT or EDxy.
- x, y, or z in EDxy or SIGNz were the same character.
- The value for LENGTH was greater than 44.
- The value for LINES was greater than 255.
- A section break field was not followed by SKIP, HEADER3, or TRAILER3.
- A statistics field was specified in HEADER1, HEADER2, or HEADER3.
- HEADER3, TRAILER3, SKIP, or PAGEHEAD was specified more than once after a section break field.

System Action: The program terminates.

Programmer Response: Correct the invalid value.

ICE232A

ICE232A ddname: SPLIT, SPLITBY, SPLIT1R OR REPEAT CANNOT BE USED FOR A REPORT

Explanation: Critical. For the OUTFIL group whose first data set is associated with ddname, a SPLIT, SPLITBY, SPLIT1R or REPEAT parameter was specified along with one or more report parameters (LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS or NODETAIL). The records of a report cannot be repeated, or split among a group of OUTFIL data sets.

System Action: The program terminates.

Programmer Response: Remove either the SPLIT, SPLITBY, SPLIT1R or REPEAT parameter or the report parameters.

ICE242A

ICE242A Z/ARCHITECTURE MODE IS REQUIRED FOR THIS FUNCTION

Explanation: Critical. The indicated function cannot be used because it requires z/Architecture mode, but you are running in ESA/390 mode. z/Architecture mode is required for the following functions:

- FL (hexadecimal floating-point) conversion to integer with the INREC, OUTREC or OUTFIL statement.
- FL (hexadecimal floating-point) conversion to integer with the DISPLAY operator.

System Action: The program terminates.

Programmer Response: A $ marks the point at which the error was detected. Remove the function or switch to z/Architecture mode.

ICE243A

ICE243A PARSED FIELD DEFINITION ERROR

Explanation: Critical. The PARSE operand of an INREC, OUTREC or OUTFIL statement contained an invalid or incorrect definition for a parsed field. One of the following errors was found:

- More than 2 digits was specified for a parsed field definition (for example, %001 instead of %1 or %01). A parsed field must start with % and must have 1 digit, 2 digits or no digits.
- FIXLEN was not specified for %nn or %n.
- FIXLEN, ABSPOS, ADDPOS, SUBPOS or PAIR was specified more than once.
- ABSPOS and ADDPOS, ABSPOS and SUBPOS, or ADDPOS and SUBPOS were specified.
- FIXLEN, ABSPOS, ADDPOS or SUBPOS was specified with a value of 0 or a value greater than 32752.
- A null string was used.
- A character or hexadecimal constant was longer than 256 bytes.
- An invalid digit or an odd number of digits was specified for a hexadecimal string.
- An invalid keyword was specified.

System Action: The program terminates.

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

ICE244A

ICE244A DUPLICATE PARSED FIELD DEFINITION

Explanation: Critical. A %nn parsed field was defined previously (for example, %03 was defined twice), a %0n parsed field was defined previously as a %n parsed field (for example, both %3 and %03 were defined), or a %n parsed field was defined previously as a %0n parsed field (for example, both %03 and %3 were defined). Each %nn (including %n and %0n) parsed field must only be defined once in all PARSE operands of the INREC, OUTREC and OUTFIL statements.

System Action: The program terminates.

Programmer Response: A $ marks the duplicate parsed field. Use unique %nn parsed fields (%00-%99).

ICE245A

ICE245A PARSED FIELD NOT DEFINED FOR USE IN THIS OVERLAY, BUILD, FIELDS OR OUTREC OPERAND

Explanation: Critical. An OVERLAY, BUILD, FIELDS or OUTREC operand, or an IFTHEN OVERLAY or IFTHEN BUILD suboperand, of an INREC, OUTREC or OUTFIL statement specified a parsed field (%nn) that was not previously defined for use with that operand or suboperand. %nn parsed fields must be defined and used as follows:

- %nn used in INREC OVERLAY/BUILD/FIELDS: The %nn parsed field must be defined in the INREC statement in a PARSE operand that precedes the OVERLAY, BUILD or FIELDS operand.
- %nn used in OUTREC OVERLAY/BUILD/FIELDS: The %nn parsed field must be defined in the OUTREC statement in a PARSE operand that precedes the OVERLAY, BUILD or FIELDS operand.
- %nn used in OUTFIL OVERLAY/BUILD/OUTREC: The %nn parsed field must be defined in the same OUTFIL statement in a PARSE operand that precedes the OVERLAY, BUILD or OUTREC operand.
- %nn used in INREC IFTHEN OVERLAY/BUILD: The %nn parsed field must be defined in the INREC statement in an IFTHEN PARSE suboperand of a WHEN=INIT clause, or in an IFTHEN PARSE suboperand of the same clause, before the OVERLAY or BUILD suboperand.
- %nn used in OUTREC IFTHEN OVERLAY/BUILD: The %nn parsed field must be defined in the OUTREC statement in an IFTHEN PARSE suboperand of a WHEN=INIT clause, or in an IFTHEN PARSE suboperand of the same clause, before the OVERLAY or BUILD suboperand.
- %nn used in OUTFIL IFTHEN OVERLAY/BUILD: The %nn parsed field must be defined in the same OUTFIL statement in an IFTHEN PARSE suboperand of a WHEN=INIT clause, or in an IFTHEN PARSE suboperand of the same clause, before the OVERLAY or BUILD suboperand.

The following are some examples of using %nn parsed fields **correctly**:

```
* %00 defined in PARSE and used in BUILD.
  INREC PARSE=(%00=(FIXLEN=10,ENDBEFR=C',')),
        BUILD=(%00)

* %01 defined in WHEN=INIT clause and used
* in WHEN=(logexp) clause.
* %02 defined in WHEN=(logexp) clause
* and used in that clause.
  OUTREC IFTHEN=(WHEN=INIT,
           PARSE=(%01=(ENDBEFR=C',',FIXLEN=8))),
         IFTHEN=(WHEN=(5,1,CH,EQ,C'A'),
              PARSE=(%02=(STARTAFT=C'(',FIXLEN=12)),
              BUILD=(%01,%02))

* %03 defined and used for OUTFIL OUT1.
* %04 defined and used for OUTFIL OUT2.
  OUTFIL FNAMES=OUT1,INCLUDE=(8,1,CH,EQ,C'A'),
         PARSE=(%03=(FIXLEN=10,ENDBEFR=C',')),
         OVERLAY=(21:%03)
  OUTFIL FNAMES=OUT2,SAVE,
         PARSE=(%04=(FIXLEN=8,ENDBEFR=C':')),
         OVERLAY=(31:%04)
```

The following are some examples of using %nn parsed fields **incorrectly**:

```
* %00 used in BUILD, but defined in PARSE
* after BUILD.
  INREC BUILD=(%00),
        PARSE=(%00=(FIXLEN=10,ENDBEFR=C','))

* %02 used in WHEN=NONE clause but not defined
* in that clause or in a WHEN=INIT clause
  OUTREC IFTHEN=(WHEN=(5,1,CH,EQ,C'A'),
            PARSE=(%02=(STARTAFT=C'(',FIXLEN=12)),
            BUILD=(%02)),
         IFTHEN=(WHEN=NONE,
            BUILD=(1,20,%02))

* %03 used in OUTFIL for OUT2, but defined
* in OUTFIL for OUT1.
  OUTFIL FNAMES=OUT1,INCLUDE=(8,1,CH,EQ,C'A'),
         PARSE=(%03=(FIXLEN=10,ENDBEFR=C',')),
         OVERLAY=(21:%03)
  OUTFIL FNAMES=OUT2,SAVE,
         OVERLAY=(21:%03)
```

System Action: The program terminates.

Programmer Response: A $ marks the incorrectly used %nn parsed field. Define the %nn parsed field correctly.

ICE272A

ICE272A SYMBOL, VALUE OR SYNTAX IS INVALID

Explanation: Critical. The SYMNAMES statement has one of the following errors:

- The symbol starts with a number (0-9) or a hyphen (-).

- The value contains an invalid parsed field. A valid parsed field must be %nn (nn is 00 to 99) or %n (n is 0 to 9).
- The symbol or value contains an invalid character. The valid characters are uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), the number sign (#), the dollar sign ($), the commercial at sign (@), the underscore(_), and the hyphen (-).
- The symbol, keyword or value is null (for example, symbol,,5,CH).
- The symbol, keyword or value is followed by or contains an invalid delimiter.
- p or m in p,m or p,m,f is 0 or greater than 32752 or contains a non-numeric character.
- q in POSITION,q is 0 or greater than 32752 or contains a non-numeric character.
- symbol in POSITION,symbol references a symbol that was not previously defined or which was previously defined without a valid position (for example, a symbol for a constant).
- n in SKIP,n is 0 or greater than 32752 or contains a non-numeric character.
- The decimal constant contains a non-numeric character other than a leading plus sign (+) or minus sign (-).
- An equal sign (=) is specified for p, m or f, but the previous position, previous length or previous format, respectively, was not established.
- f in p,m,f is not a valid format. The valid formats are AC, AQ, ASL, AST, BI, CH, CLO, CSF, CSL, CST, CTO, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, D1, D2, FI, FL FS, LS, OL, OT, PD, PD0, SFF, SS, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, TM4, TS, UFF, Y2B, Y2C, Y2D, Y2DP, Y2P, Y2PP, Y2S, Y2T, Y2TP, Y2U, Y2UP, Y2V, Y2VP, Y2W, Y2WP, Y2X, Y2XP, Y2Y, Y2YP, Y2Z and ZD, and lowercase or mixed case variations
- x in ALIGN,x is not a valid alignment. The valid alignments are H, F, D, h, f and d.
- The character constant, system symbol constant, hexadecimal constant or bit constant does not have an ending apostrophe after the string.
- The hexadecimal constant is null (X'') or contains an odd number of digits (for example, X'123').
- The hexadecimal constant contains an invalid character. The valid characters are 0-9, A-F and a-f.
- The bit constant is null (B'') or contains a number of bits that is not a multiple of 8 (for example, B'1010').
- The bit constant contains an invalid character. The valid characters are . (period), 0 and 1.

System Action: The program terminates.

Programmer Response: Correct the symbol, value or syntax error.

ICE276A

ICE276A RESERVED WORD - NOT ALLOWED FOR SYMBOL

Explanation: Critical. The SYMNAMES statement specifies a DFSORT/ICETOOL reserved word for the symbol. Reserved words cannot be used for symbols. The reserved words are as follows (uppercase only as shown): A, AC, ADD, ALL, AND, AQ, ASL, AST, BI, CH, CLO, COPY, COUNT, COUNT15, CSF, CSL, CST, CTO, D, DATE, DATE1, DATE1..., DATE2, DATE2..., DATE3, DATE3..., DATE4, DC1, DC2, DC3, DE1, DE2, DE3, DIV, DT1, DT2, DT3, D1, D2, E, F, FI, FL, FS, H, HEX, LS, MAX, MIN, MOD, MUL, Mn, Mnn, NONE, NUM, OL, OR, OT, PAGE, PAGEHEAD, PD, PDC, PDF, PD0, SEQNUM, SFF, SS, SUB, SUBCOUNT, SUBCOUNT15, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TIME, TIME1, TIME1P, TIME2, TIME2P, TIME3, TIME3P, TM1, TM2, TM3, TM4, TS, UFF, VALCNT, VLEN, X, Y2x, Y2xx, Z, ZD, ZDC, and ZDF, where n is 0-9 and x is any character.

System Action: The program terminates.

Programmer Response: Use a symbol that is not one of the reserved words, such as a lowercase or mixed case version of the word being used. For example, you could use Valcnt (which is not a reserved word) instead of VALCNT (which is).

ICE619A

ICE619A INVALID LENGTH, FORMAT, OR COMBINATION FOR operator OPERATION

Explanation: Critical. One of the following conditions was detected in the parameters of an ON or BREAK operand:

- The format was invalid. Example: ON(10,2,CST) or BREAK(10,2,FL)
- The format was not allowed for this operator. Example: VERIFY ON(10,2,BI)
- The length was not within the range allowed for the format and operator. Example: STATS ON(10,9,BI)
- ON(VLEN) was specified for a VERIFY operator. Example: VERIFY ON(VLEN)
- ON(NUM) was specified for an operator other than DISPLAY. Example: STATS ON(NUM)
- ON(VALCNT) was specified for an operator other than OCCUR. Example: DISPLAY ON(VALCNT)
- ON(p,m,HEX) was specified for an operator other than DISPLAY or OCCUR. Example: UNIQUE ON(5,4,HEX)
- The length was not within the range allowed for ON(p,m,HEX). Example: ON(5,1001,HEX)
- BREAK(p,m,HEX) was specified. Example: BREAK(5,4,HEX)

System Action: The program terminates.

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

ICE637A

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

Explanation: Critical. 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 line (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.

System Action: The program terminates.

Programmer Response: If m is less than 2048, 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, 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).

ICE652A

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

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

System Action: The program terminates.

Programmer Response: If you want to reformat the output records produced by this SELECT or SPLICE operator, use one or more OUTFIL statements as explained in *z/OS DFSORT Application Programming Guide*, instead of an OUTREC statement.