IBM i
7.2

Programming
DDS for Physical and Logical files



Note

Before using this information and the product it supports, read the information in “Notices” on page 89.

This edition applies to IBM i 7.2 (product number 5770-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

This document may contain references to Licensed Internal Code. Licensed Internal Code is Machine Code and is licensed to you under the terms of the IBM License Agreement for Machine Code.

© Copyright International Business Machines Corporation 1999, 2013.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

DDS for physical and logical files....1
PDF file for DDS for Physical and Logical files....1
Defining physical and logical files using DDS....1
Defining a physical file using DDS....1
Defining a logical file using DDS....2
Simple- and multiple-format logical files in DDS....2
Join logical files in DDS....3
Specifying record formats in a logical file in DDS....3
Positional entries for physical and logical files (positions 1 through 44)....4
Sequence number for physical and logical files (positions 1 through 5)....5
Form type for physical and logical files (position 6)....5
Comment for physical and logical files (position 7)....5
Condition for physical and logical files (positions 8 through 16)....5
Type of name or specification for physical and logical files (position 17)....5
Reserved for physical and logical files (position 18)....6
Name for physical and logical files (positions 19 through 28)....6
Record format....6
Field name....8
Key field name....8
Select/omit field name....18
Reference for physical and logical files (position 29)....20
Length for physical and logical files (positions 30 through 34)....21
Data type for physical and logical files (position 35)....23
Conversion of one numeric data type to another in a DDS file....26
Conversion between zoned decimal and character or hexadecimal in a DDS file....26
Conversion of a field from floating point to packed decimal, zoned decimal, or binary in DDS....26
Conversion of data types when concatenating fields in DDS....26
Conversion of data types when substringing fields in DDS....27
Decimal positions for physical and logical files (positions 36 and 37)....27
Usage for physical and logical files (position 38)....28
Location for physical and logical files (positions 39 through 44)....30
Keyword entries for physical and logical files (positions 45 through 80)....30
ABSVAL (Absolute Value) keyword for physical and logical files....31
ALIAS (Alternative Name) keyword for physical and logical files....32
ALL (All) keyword—logical files only....32
ALTSEQ (Alternative Collating Sequence) keyword for physical and logical files....33
ALWNULL (Allow Null Value) keyword—physical files only....34
CCSID (Coded Character Set Identifier) keyword for physical and logical files....34
CHECK (Check) keyword for physical and logical files....36
CHKMSGID (Check Message Identifier) keyword for physical and logical files....37
CMP (Comparison) keyword for physical and logical files....37
COLHDG (Column Heading) keyword for physical and logical files....37
COMP (Comparison) keyword for physical and logical files....38
Specifying COMP at the field level....39
Specifying COMP at the select/omit-field level....40
CONCAT (Concatenate) keyword—logical files only....41
DATFMT (Date Format) keyword for physical and logical files....43
DATSEP (Date Separator) keyword for physical and logical files....45
DESCEND (Descend) keyword for physical and logical files....46
DFT (Default) keyword—physical files only....46

DIGIT (Digit) keyword for physical and logical files ... 48
DYNSLT (Dynamic Select) keyword—logical files only ... 49
EDTCDE (Edit Code) and EDTWRD (Edit Word) keywords for physical and logical files ... 51
FCFO (First-Changed First-Out) keyword for physical and logical files ... 52
FIFO (First-In First-Out) keyword for physical and logical files ... 53
FLTPCN (Floating-Point Precision) keyword for physical and logical files ... 53
FORMAT (Format) keyword for physical and logical files ... 54
JDFTVAL (Join Default Values) keyword—join logical files only ... 54
JDUPSEQ (Join Duplicate Sequence) keyword—join logical files only ... 55
JFILE (Joined Files) keyword—join logical files only ... 57
JFLD (Joined Fields) keyword—join logical files only ... 58
JOIN (Join) keyword—join logical files only ... 60
JREF (Join Reference) keyword—join logical files only ... 62
LIFO (Last-In First-Out) keyword for physical and logical files ... 63
NOALTSEQ (No Alternative Collating Sequence) keyword for physical and logical files ... 63
PFILE (Physical File) keyword—logical files only ... 64
RANGE (Range) keyword for physical and logical files ... 65
Specifying RANGE at the field level ... 66
Specifying RANGE at the select/omit-field level ... 66
REF (Reference) keyword—physical files only ... 67
REFACCPTH (Reference Access Path Definition) keyword—logical files only ... 67
REFFLD (Referenced Field) keyword—physical files only ... 68
REFSHIFT (Reference Shift) keyword for physical and logical files ... 69
RENAME (Rename) keyword—logical files only ... 70
SIGNED (Signed) keyword for physical and logical files ... 71
SST (Substring) keyword—logical files only ... 72
TEXT (Text) keyword for physical and logical files ... 74
TIMFMT (Time Format) keyword for physical and logical files ... 74
TIMSEP (Time Separator) keyword for physical and logical files ... 75
TRNTBL (Translation Table) keyword—logical files only ... 76
UNIQUE (Unique) keyword for physical and logical files ... 77
UNSIGNED (Unsigned) keyword for physical and logical files ... 78
VALUES (Values) keyword for physical and logical files ... 79
Specifying VALUES at the field level ... 79
Specifying VALUES at the select/omit-field level ... 80
VARLEN (Variable-Length Field) keyword for physical and logical files ... 80
ZONE (Zone) keyword for physical and logical files ... 81
Unicode considerations for database files ... 82
Unicode considerations for database files: Length (positions 30 through 34) ... 82
Unicode considerations for database files: Data type (position 35) ... 83
Unicode considerations for database files: Decimal positions (positions 36 and 37) ... 83
Unicode considerations for database files: Keyword considerations (positions 45 through 80) ... 83
DBCS considerations for database files ... 84
Positional entry considerations for database files that use DBCS ... 84
Length (positions 30 through 34) ... 84
Data type (position 35) ... 85
Decimal (positions 36 and 37) ... 85
Keyword considerations for database files that use DBCS ... 85
CONCAT (Concatenate) keyword ... 86
Additional considerations for describing database files that contain DBCS data ... 87

Notices ... 89
Programming interface information ... 90
Trademarks ... 90
Terms and conditions ... 91

Index ... 93

DDS for physical and logical files

You can use data description specifications (DDS) to define physical and logical database files.

This topic collection provides the information you need to code the positional and keyword entries that define physical and logical files.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 88.

PDF file for DDS for Physical and Logical files

You can view and print a PDF file of this information.

To view or download the PDF version of this document, select DDS for Physical and Logical files.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html).

Defining physical and logical files using DDS

You can use data description specifications (DDS) to define physical and logical database files.

Defining a physical file using DDS

You use file-level, record-level, field-level, and key-field-level entries to define a physical file with data description specifications (DDS).

A physical file can contain only one record format. To define a physical file, follow these steps:

1. Specify the record format in either of the following two ways:
   - Define a new record format.

     Specify field and key-field specifications as required for the new record format.
   - Share an existing record format.

     Use the FORMAT keyword to specify that the operating system is to use a previously defined record format from another physical file. When the FORMAT keyword is used, you must specify key-field level specifications again (if a keyed access path is required) even if they were specified on the existing record format.
2. Specify the entries in the following order to define a physical file:
   a. File-level entries
   b. Record-level entries
   c. Field-level entries

© Copyright IBM Corp. 1999, 2013

d. Key field-level entries

Note: The file name is specified through the Create Physical File (CRTPF) command, not through DDS.

The maximum number of fields in a record format is 8000. If any of the fields in the record format are date, time, timestamp, variable length, or allows the null value, then the actual maximum number of fields can be less than 8000. The maximum number of fields can vary depending on the number of fields and combinations of fields that occur within the record format. The maximum number of bytes in a record format is 32 766 if variable length fields are not included and 32 740 if variable length fields are included.

Related information

Rules for DDS keywords and parameter values

Defining a logical file using DDS

A logical file determines how data records are selected and defined when read by an application program. A logical file can be a simple, multiple format, or join logical file.

A simple logical file contains one record format and has one file specified on the PFILE keyword. A multiple format logical file either contains more than one record format or has more than one file specified on the PFILE keyword. A join logical file contains one record format and has up to 256 files specified on the JFILE keyword.

Simple- and multiple-format logical files in DDS

You must specify the PFILE keyword at the record level for simple- and multiple-format logical files. In a multiple-format logical file, a record format can use only the fields common to all the physical files specified on the PFILE keyword for that record format.

Specify the entries in the following order to define a simple- or multiple-format logical file:

1. Optional: File-level entries
2. Record-level entries
3. Optional: Field-level entries
4. Optional: Key field-level entries
5. Optional: Select/omit field-level entries

Repeat steps 2 through 5 for each record format in the file.

The following example shows the code for a multiple-format logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A* LOGICAL FILE EXAMPLE
00020A* INVENTORY FORMAT
00030A          R INVFMT                    PFILE(INVENTORY)
00040A          K ITEM
00050A*
00060A* ORDER FORMAT
00070A          R ORDFMT                    PFILE(ORDER)
00080A                                      TEXT('ORDER ANALYSIS')
00090A            ITEM
00100A            ORDER         10
00110A            SUPPLY        +2
00120A            SHPDAT                    CONCAT(SHPMO SHPDA SHPYR)
00130A            QTY            5P         RENAME(QTYDUE)
00140A          K ITEM
00150A          K SHPYR
00160A          K SHPMO
00170A          K SHPDA
00180A          O QTYDUE                    CMP(LT 1)
00190A*
00200A* ACCOUNTING FORMAT
00210A          R ACTFMT                    PFILE(ACCOUNTS)
00220A                                      FORMAT(ACCOUNTL)
00230A          K ITEM
     A
```

Related reference

Positional entries for physical and logical files (positions 1 through 44)
The first 44 positions of the DDS form are called positional entries.

Name for physical and logical files (positions 19 through 28)
You use these positions to specify record or field names.

Length for physical and logical files (positions 30 through 34)
You use these positions to specify the length of a physical or logical file field.

Data type for physical and logical files (position 35)
For a physical file, you use this position to specify the data type of the field within the database. You specify data type in a logical file only to override or change the data type of the corresponding field in the physical file on which this logical file is based.

Join logical files in DDS

Join logical files combine different fields from more than one physical file into a single record. You must specify the JFILE keyword at the record level for join logical files.

Specify the entries in the following order to define a join logical file:

Note: Because only one record format is allowed in a join logical file, specify these entries only once.

1. Optional: Specify file-level entries
2. Specify record-level entries
3. Specify join-level entries
4. Specify field-level entries
5. Optional: Specify key field-level entries
6. Optional: Specify select/omit-field level entries

The following example shows the code for a join logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A* Joins fields from two physical files into one record format
00020A          R RECORD1                   JFILE(PF1 PF2)
00030A          J                           JOIN(PF1 PF2)
00040A                                      JFLD(NAME NAME)
00050A            NAME                      JREF(1)
00060A            ADDR
00070A            PHONE
     A
```

Related reference

Positional entries for physical and logical files (positions 1 through 44)
The first 44 positions of the DDS form are called positional entries.

Name for physical and logical files (positions 19 through 28)
You use these positions to specify record or field names.

Related information

Rules for DDS keywords and parameter values

Specifying record formats in a logical file in DDS

If there is more than one record format specified in a logical file, you must specify the PFILE keyword for each record format.

There are three ways to specify the fields in a record format:

- Specify the record format name and the PFILE keyword.
- Specify the record format name, the PFILE or JFILE keyword, and at least one individual field.
- Specify the record format name, the PFILE keyword, and the FORMAT keyword.

For each of the three ways to specify fields in a record format, you can have one of the following access path specifications:

- Specify no key fields (arrival sequence access path).

  You cannot specify select/omit fields unless you specify the DYNSLT keyword. You can specify only one record format with one physical file on the PFILE keyword for the logical file.

- Specify one or more key fields (keyed sequence access path).

  If you specify more than one record format in the logical file, each record format must have at least one key field specified. You can specify select/omit fields for any of the record formats in the file.

- Specify the REFACCPTH keyword (keyed sequence access path).

  The access path information from another physical or logical file is copied into the file that you are defining.

The maximum number of fields in a record format is 8000. If any of the fields in the record format are date, time, timestamp, variable length, or allows the null value, then the actual maximum number of fields can be less than 8000. The maximum number of fields can vary depending on the number of fields and combinations of fields that occur within the record format. The maximum number of bytes in a record format is 32 766 if variable length fields are not included and 32 740 if variable length fields are included.

Related reference

Length for physical and logical files (positions 30 through 34)
You use these positions to specify the length of a physical or logical file field.

Positional entries for physical and logical files (positions 1 through 44)

The first 44 positions of the DDS form are called positional entries.

The following example shows some positional entries for physical files.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A* PHYSICAL FILE CODING EXAMPLE
00020A                                     REF(INVENCTL/INVENTORY)
00030A                                     UNIQUE
00040A          R ORDFMT                   TEXT('Format for Purchase Orders')
00050A            ORDNBR          7  0     COLHDG('Order' 'Number')
00060A            ITMNBR    R   10
00070A            SUPNBR    R   +2         REFFLD(SUPID SUPLIB/SUPMST)
00080A            QTYORD         5B
00090A          K ORDNBR
00100A          K ITMNBR                   ABSVAL
     A
```

Figure 1. Physical file example

Related tasks

Simple- and multiple-format logical files in DDS
You must specify the PFILE keyword at the record level for simple- and multiple-format logical files. In a multiple-format logical file, a record format can use only the fields common to all the physical files specified on the PFILE keyword for that record format.

Join logical files in DDS

Join logical files combine different fields from more than one physical file into a single record. You must specify the JFILE keyword at the record level for join logical files.

Sequence number for physical and logical files (positions 1 through 5)

You can use the first five positions to indicate the sequence number for each line on the form. The sequence number is optional and is used for documentation purposes only.

Form type for physical and logical files (position 6)

You type an A in this position to designate this as a DDS form. The form type is optional and is for documentation purposes only.

Comment for physical and logical files (position 7)

You type an asterisk (*) in this position to identify this line as a comment, and then use positions 8 through 80 for comment text.

A blank line (no characters specified in positions 7 through 80) is treated as a comment. Comments can appear anywhere in DDS and are kept only in the source file. Comments are printed on the source computer printout but not on the expanded source computer printout.

Condition for physical and logical files (positions 8 through 16)

These positions do not apply to physical or logical files. Leave these positions blank unless you use them for comment text.

Type of name or specification for physical and logical files (position 17)

For physical files, type a value in this position to identify the type of name. For logical files, type a value to identify the type of specification. If you specify a name type, the name is specified in positions 19 through 28.

The valid entries for physical files are:

Entry
Meaning

R
Record format name

Blank
Field name

K
Key field name

Note: Specify only one R because a physical file can contain only one record format.

The valid entries for logical files are:

Entry
Meaning

R
Record format name

J
Join specification

Blank
Field name or select/omit AND condition

K
Key field name

S
Select field name

O
Omit field name

Related reference
Name for physical and logical files (positions 19 through 28)
You use these positions to specify record or field names.

JOIN (Join) keyword—join logical files only
You can use this join-level keyword to identify which pair of files are joined by the join specification in which you specify this keyword.

Reserved for physical and logical files (position 18)

This position does not apply to any file type. Leave this position blank unless you use it for comment text.

Name for physical and logical files (positions 19 through 28)

You use these positions to specify record or field names.

You can specify the names of the following items:

- The record format for this physical file or formats for this logical file
- The field name or field names that make up the record format (unless you specify the FORMAT or PFILE keyword at the record level)
- The field or fields used as key fields
- For logical files, the field or fields to be used for select/omit specifications

Note: The file name is specified through the Create Physical File (CRTPF) command, not in the DDS.

Names must begin in position 19.

You must specify the name type in position 17, unless you are specifying a field name or select/omit AND condition.

Related tasks
Simple- and multiple-format logical files in DDS
You must specify the PFILE keyword at the record level for simple- and multiple-format logical files. In a multiple-format logical file, a record format can use only the fields common to all the physical files specified on the PFILE keyword for that record format.

Join logical files in DDS
Join logical files combine different fields from more than one physical file into a single record. You must specify the JFILE keyword at the record level for join logical files.

Related reference
Type of name or specification for physical and logical files (position 17)
For physical files, type a value in this position to identify the type of name. For logical files, type a value to identify the type of specification. If you specify a name type, the name is specified in positions 19 through 28.

Related information
Rules for DDS keywords and parameter values

Record format

When you specify R in position 17, the name specified in positions 19 through 28 is a record format name.

Record format for physical files

Only one record format name is allowed for a physical file. Specify the record format name in one of the following two ways:

- As the name of a new record format with field names specified in this physical file.

  The name of the record format can be the same as the file name specified in the Create Physical File (CRTPF) command. However, a warning message appears if the names are not unique because some high-level language processors do not allow record format and file names to be the same. Report Program Generator (RPG) is such a high-level language. The record format name and field names do not need to be unique to the system; the same names can exist in another file.
- As the name of a record format previously defined in another physical file.

  The FORMAT keyword must be specified. Field names and attributes are not specified.

Record format for simple- and multiple-format logical files

You can specify more than one record format name. However, each name must be unique within the file. See the appropriate high-level language manual for exceptions.

Specify the record format name in one of the following three ways:

- As the record format name in the first physical file specified on the PFILE keyword.

  This is required if you do not specify the FORMAT keyword and do not identify individual fields by naming them in this record format.
- As the name of a new record format with field names specified in this logical file.

  Every field must be identified by name. No unnamed physical file fields are part of this logical file record format. Physical file fields that are parameters of RENAME and CONCAT keywords are part of the logical file record format. Physical file fields that are parameters of SST keywords are not part of the logical file record format unless specified elsewhere.
- As the name of a record format previously described in a physical or logical file.

  Field names and attributes are not specified and the FORMAT keyword must be specified.

The record format name can be the same as the file name specified in the create file command. However, a warning message is sent if the names are not unique. Some high-level language processors, such as RPG, do not allow record format and file names to be the same.

Use the PFILE keyword in conjunction with the record format name to specify the physical files with which the record format is to be associated. A record format can have more than one physical file specified on the PFILE keyword. If no fields are defined and the FORMAT keyword is not specified, the format of the first file specified in the PFILE keyword is used as the format for all the physical files. (This format is used for field attribute references and attribute and name checking.)

Join logical files

Only one record format name can be specified. Specify the record format name as the name of a new record format with field names specified in this logical file. Every field in the record format for a join logical file must be identified by the name in positions 19 through 28. Physical file fields that are parameters of the RENAME, CONCAT, and SST keywords are part of the logical file record format only if you specify the field names elsewhere in the record format.

The JFILE keyword is required at the record level. It specifies the physical files that the record format joins.

Related reference

FORMAT (Format) keyword for physical and logical files

You can use this record-level keyword to specify that this record format is to share the field specifications for a previously defined record format. The name of the record format you are defining must be the name of the previously defined record format.

Field name

When position 17 is left blank, the name specified in positions 19 through 28 is a field name.

You cannot specify field names if you specify the FORMAT keyword.

Physical files require that each field be named. These names must be unique within the record format. The field names appear in the physical buffer in the same order as they are specified in the DDS.

If you are describing a simple or multiple format logical file, you can use the record format as it exists in the physical file on which this logical file is based, and you do not need to specify field names.

If you do not use the record format as it exists in the physical file, you must name each field specified in a logical file. In a simple or multiple format logical file, each field name must be unique within the record format and must correspond to a field in the physical file record format. The field name order is the order in which the fields appear to programs using the logical file.

The name you give to a field in a logical file record format is typically the same as the corresponding field name in the physical file record format. If different, the two names must be equated by using the RENAME keyword. A field in a logical file record format can also represent the concatenation of two or more fields from the physical file. The SST keyword can also be used to describe a substring of a field from the physical file in the logical file format.

Note: The sequence in which the field names are specified in the logical file is important. If the same physical field is specified more than once in a record format in the logical file (by using either RENAME or CONCAT), the sequence in which the fields are specified in the logical file is the sequence that the data is moved to the physical file. Thus, the value of the field specified in the logical file the last time is the value in the physical record.

Related reference

CONCAT (Concatenate) keyword—logical files only

You can use this field-level keyword to combine two or more fields from the physical-file record format into one field in the logical-file record format you are defining. The name of this concatenated field must appear in positions 19 through 28.

Key field name

When you specify K in position 17, the name specified in positions 19 through 28 is a key field name.

It must be one of the field names within the physical file record format. The contents of this field are used to sequence the records for retrieval from the database. Specifying a key is optional. If no key field is specified, the default sequence is arrival sequence (the order that the records were put into the file).

Use key fields (and optionally, select/omit fields) to define a keyed sequence access path for record formats in the logical file member. The logical file member includes the physical file members specified on the DTAMBRS parameter on the Create Logical File (CRTLF) or Add Logical File Member (ADDLFM) command.

You can change the sequence of records as they are read from the file by specifying a sequencing keyword. The sequencing keywords are ALTSEQ, NOALTSEQ, SIGNED, UNSIGNED, ABSVAL, ZONE, DIGIT, DESCEND, FCFO, FIFO, and LIFO.

When you do not specify any sequencing keyword for a key field, the default sequence for that key field is ascending order. The default for character, hexadecimal, date, time, and timestamp fields is the UNSIGNED attribute. The default for numeric fields is the SIGNED attribute, except for zoned decimal fields (S specified in position 35) in the following cases:

- When you specify ALTSEQ at the file level, all zoned decimal key fields in the file are set to UNSIGNED as default.
- When you specify DIGIT or ZONE for a zoned decimal key field, the field is set to UNSIGNED as default.

If you specify more than one record format for a logical file or more than one physical file for the PFILE keyword, you must specify at least one key field for all record formats of that logical file.

A key can have more than one key field. This is called a *composite key*. In a composite key, specify the key field names in the order of importance (major to minor), and specify each key field name on a separate line.

Figure 2 on page 9 shows a multiple format logical file with two record formats, one of which uses a composite key. In this example, RECORD1 has a single key field, FIELD1. RECORD2 has a composite key that includes FIELD4 and FIELD5.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PF1)
00020A            FIELD1
00030A            FIELD2
00040A            FIELD3
00050A          K FIELD1
00060A*
00070A          R RECORD2                   PFILE(PF2)
00080A            FIELD4
00090A            FIELD5
00100A          K FIELD4
00110A          K FIELD5
     A
```

Figure 2. Specifying a multiple-format logical file with two record formats

If you do not specify a key field for a logical file, the file you are defining has an arrival sequence access path.

The number of fields that make up a key is restricted to 120. The total key length cannot exceed 32 768 bytes. (If the FCFO keyword is specified, the total key length cannot exceed 32 763 bytes. In a DDM environment, the key length is limited to 12 000.) The total key length includes the length of each key field. If any of the key fields allow the null value, add 1 byte for each key field that allows the null value. The operating system uses the extra byte to determine whether the key contains the null value. If any of the key fields is variable length, add 2 bytes for each variable-length key field. The operating system uses the extra 2 bytes to store the allocated length of the field.

When you specify more than one record format in a logical file, an additional byte for the first *NONE key field position is required. An additional byte might also be required for each additional key field position. The operating system uses the extra bytes when records from different physical files have duplicate key values.

For example, suppose a key consists of fields named FIELDA, FIELDB, and FIELDC (in that order). The DDS appears as shown in Figure 3 on page 9.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A* SAMPLE COMPOSITE KEY (PHYSICAL FILE)
00020A          R RECORD
00030A            FIELDA         3  0
00040A            FIELDB         3  0
00050A            FIELDC         3  0
00060A            FIELDD         3  0
00070A          K FIELDA
00080A          K FIELDB
00090A          K FIELDC
     A
```

Note: Lines 00070 to 00090 make up the composite key.

Figure 3. Composite key

The records are sequenced in the following order:

1. They are sequenced according to the contents of FIELDA.
2. If two or more records with the same value in FIELDA exist, the operating system sequences those records according to the values in FIELDB.

3. If two or more of those records have the same value in both FIELDA and FIELDB, they are sequenced according to the values in FIELDC.

Consider the following file:

Record	FIELDA	FIELDB	FIELDC
1	333	99	67
2	444	10	45
3	222	34	23
4	222	12	01
5	222	23	45
6	111	06	89
7	222	23	67

Assuming ascending sequencing for all fields, the records are retrieved in this order:

Record	FIELDA	FIELDB	FIELDC
6	111	06	89
4	222	12	01
5	222	23	45
7	222	23	67
3	222	34	23
1	333	99	67
2	444	10	45

The following information applies:

- Because records 3, 4, 5, and 7 have the same contents in FIELDA, FIELDB becomes the determining field.
- Within those four records, 5 and 7 have the same values in FIELDB. For these two records, FIELDC becomes the determining field.
- If FIELDC also contains duplicate values, the records are retrieved in first-in first-out (FIFO), last-in first-out (LIFO), or first-changed first-out (FCFO) order. To guarantee the order, specify the FIFO keyword, the LIFO keyword, or the FCFO keyword. Specify the UNIQUE keyword to prevent duplicate key values.

See SIGNED (Signed) keyword for physical and logical files for an example that includes a key field with negative (-) contents.

Special restrictions apply to key field specifications when either FILETYPE(*SRC) is used on the Create Physical File (CRTPF) command or for the Create Source Physical File (CRTSRCPF) command.

For logical files, the following rules apply to fields that you specify as key fields:

- For simple and multiple format logical files, the following search order is used to match key field names with defined fields:

  1. Fields specified in DDS positions 19 through 28
  2. Fields specified as parameters on the CONCAT or RENAME keyword

  If the field name is specified more than once, the first occurrence is used.

  The field name on a CONCAT or RENAME keyword and the associated field name in positions 19 through 28 cannot both be specified as key fields.

The parameter name on the SST keyword is not valid as a key field unless it is defined elsewhere in the logical file format.

- For join logical files, the key field name you specify must be specified at the field level in positions 19 through 28 and must be a field described in the primary file (the first physical file specified on the JFILE keyword).

  Note: If you specify a field as a parameter value on the CONCAT, RENAME, or SST keyword, but do not specify the field in positions 19 through 28 of the join logical file, you cannot specify the field as a key field.

If you are concatenating numeric with either character or hexadecimal, you cannot specify the numeric fields as key fields. If you are concatenating zoned decimal and fields of any other numeric data type, you cannot specify the fields of the other data types as key fields.

Figure 4 on page 11 illustrates which concatenated fields can and cannot be used as key fields.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PF1)
00020A            FLD1
00030A            FLD2
00040A            Z                         CONCAT(ZFLD PFLD)
00050A            A                         CONCAT(AFLD NFLD)
00060A          K ZFLD
00070A          K AFLD
     A
```

Figure 4. Correct and incorrect concatenated fields

In physical file PF1, ZFLD is zoned decimal and PFLD is packed decimal. Therefore Z is zoned decimal, and PFLD cannot be used as a key field. ZFLD and Z can be used as key fields but not in the same record format.

In physical file PF1, AFLD is a character field and NFLD is a numeric field. Therefore A is character, and NFLD cannot be used as a key field. AFLD and A can be used as key fields but not in the same record format.

Related reference

CONCAT (Concatenate) keyword—logical files only
You can use this field-level keyword to combine two or more fields from the physical-file record format into one field in the logical-file record format you are defining. The name of this concatenated field must appear in positions 19 through 28.

SIGNED (Signed) keyword for physical and logical files
If this key field-level keyword is in effect, when sequencing the values associated with this numeric key field, the operating system considers the signs of the values (negative versus positive values).

DDS access path keywords
You can specify one or more access path keywords to affect the way the operating system builds and uses key values.

The access path keywords are:

File level	Key field level
ALTSEQ	DESCEND
FCFO	DIGIT
FIFO	SIGNED
LIFO	UNSIGNED
REFACCPTH	ZONE
UNIQUE	

Different key fields within a composite key can have different access path keywords.

DDS logical files with more than one record format
When you specify more than one record format in a logical file, you must specify at least one key field for every record format in the logical file.

It is not necessary to specify the same number of key fields in each key. Also, key fields specified in one record format must have the same field attributes and access path keywords as the corresponding key fields in other record formats in the same logical file. Variable-length key fields are not be allowed to align with a fixed-length key field, even if the field types and lengths are the same.

A key is required for every record format so that the logical file members can have a single access path sequencing records of each record format. When records are returned from the various members of the physical file on which the logical file is based, they are merged according to the values of the key fields in the access path for the logical file member.

When records of a logical file member are sequenced, the operating system builds a key value for each record by concatenating the values in its key fields. The key value is then used to build the access path for use by your program.

Each key field in a composite key has a key position. The first key field specified is in position 1; the second key field specified is in position 2, and so on. During I/O operations to a logical file, the IBM® i program compares the key values of the records written to or read from the database. When you create a logical file that has more than one record format (with or without different key fields specified), the operating system performs key position attribute checking. For key position attribute checking to succeed, key fields of different record formats that are in the same key positions must have the same data type, length, decimal positions, and access path keywords specified at the key field level. This ensures a meaningful record sequence from the comparisons made during an I/O operation.

Floating-point fields used as key fields must have the same data type and precision but need not have the same length and decimal positions.

In the following example, FIELD1, FLD1, and F1 must have the same attributes, and FIELD2, FLD2, and F2 must have the same attributes. FIELD1, FLD1, and F1 are in key position 1; FIELD2, FLD2, and F2 are in key position 2. One record format can have more key fields than another, and the additional fields do not need key position attribute checking. FLD3 is such a field.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PF1)
00020A            FIELD1         5  0
00030A            FIELD2        10
00040A            FIELD3        10
00050A          K FIELD1
00060A          K FIELD2                    DESCEND
00070A*
00080A          R RECORD2                   PFILE(PF2)
00090A            FLD1           5  0
00100A            FLD2          10
00110A            FLD3          20
00120A          K FLD1
00130A          K FLD2                      DESCEND
00140A          K FLD3
     A*
     A          R RECORD3                   PFILE(PF3)
     A            F1             5  0
     A            F2            10
     A            F3            30
     A          K F1
     A          K F2                        DESCEND
     A
     A
```

Figure 5. Key field attribute checking

In the example in “Simple- and multiple-format logical files in DDS” on page 2, fields named ITEM are specified in each key. For record formats INVFMT and ACTFMT, ITEM is the only key field specified. For record format ORDFMT, a composite key is specified. This composite key includes ITEM, SHPYR, SHPMO, and SHPDA. Each of the fields used in a key must also exist at the field level. Therefore, ITEM must exist in the record format for the physical file INVENTORY so that it can be copied into this logical file for INVFMT.

Also, ITEM must exist in the record format for the logical file ACCOUNTL so that it can be copied into this logical file for ACTFMT. ITEM must also exist in physical file ACCOUNTS.

*Using *NONE in the key field when creating a DDS file*
Key fields having the same key position should not be compared under two conditions.

The two conditions are:

- The key fields do not have the same field attributes (data type, length, decimal positions, or access path keywords at the field level).
- The key fields have the same attributes, but you do not want them to be merged and sequenced together.

To avoid unwanted comparisons between key fields, specify *NONE in place of one of them and move the displaced key field to the next key position. The operating system compares the values of key positions before and after *NONE, but retrieves the affected records in the order in which the record formats are specified in the DDS for the logical file.

You can specify *NONE two or more times on the following lines to displace a key field to a key position for which a comparison of key field attributes is relevant to your application.

Figure 6 on page 13 shows *NONE as the key field.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RECORD1                   PFILE(PF1)
     A            FIELD1          6A
     A            FIELD2          4A
     A            FIELD3         10A
     A          K FIELD1
     A          K FIELD2
     A          K FIELD3
     A
     A          R RECORD2                   PFILE(PF2)
     A            FLD1            3A
     A            FLD2            4A
     A            FLD3           12A
     A          K *NONE
     A          K FLD2
     A
     A          R RECORD3                   PFILE(PF3)
     A            F1              6A
     A            F2              4A
     A            F3             10A
     A          K F1
     A          K F2
     A          K F3
     A
```

*Figure 6. Specifying *NONE as the key field*

In Figure 6 on page 13, the attributes for FIELD2, FLD2, and F2, must be identical. Because you specified *NONE for the first key field of the second record, then FIELD1 and F1 (first key field of the first record and first key field of the third record) must have identical attributes. FIELD3 and F3 must also have identical attributes; there is no corresponding field in the second record format.

Related reference
Example 1: Specifying the key field
In this example, a logical file views records of two physical files through two different record formats: CLSHST (class history) and JOBHST (job history).

Example 2: Specifying the key field

In this example, a logical file views the same two physical files as in Example 1, but the second record format in the logical file has *NONE specified in key position 2.

Example 1: Specifying the key field
In this example, a logical file views records of two physical files through two different record formats: CLSHST (class history) and JOBHST (job history).

In the logical file, the records from the two physical files can be merged together and sequenced by employee identification number (EMPNBR) by specifying EMPNBR in key position 1.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R CLSHST                    PFILE(CLSHSTP)
00020A          K EMPNBR  1
00030A          K CLSDTE  2
00040A*
00050A          R JOBHST                    PFILE(JOBHSTP)
00060A          K EMPNBR  1
00070A          K JOBDTE  2
     A
```

Figure 7. Specifying the key field

Record format	Key positions	
	1	2
CLSHST	EMPNBR	CLSDTE
JOBHST	EMPNBR	JOBDTE

All records that have the same key value for EMPNBR pertain to the same employee. To merge and sequence all records for a given employee into a single history of classes and job assignments, specify CLSDTE (date of class) and JOBDTE (date of job assignment) in key position 2 for the two record formats, as shown in the Using *NONE in the key field when creating a DDS file topic.

Suppose that the job assignment dates and class dates are the dates (month/year) that the class or assignment started. Records for three students are retrieved in the following order:

EMPNBR	CLSDTE	JOBDTE	Description
1005	3/79		Completed class
1005	4/79		Left to begin new job
1005		4/79	Completed job
1005	6/79		Completed class
1006		1/79	Completed job
1006		2/79	Completed job
1006	3/79		Completed class
1006	5/79		Transferred to new location
1007		1/79	Completed job
1007		4/79	Completed job
1007		7/79	Completed job
1007	8/79		Left because of illness

The above report provides a continuous history for each student.

Related concepts
Using *NONE in the key field when creating a DDS file

Key fields having the same key position should not be compared under two conditions.

Example 2: Specifying the key field
In this example, a logical file views the same two physical files as in Example 1, but the second record format in the logical file has *NONE specified in key position 2.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R CLSHST                    PFILE(CLSHSTP)
00020A          K EMPNBR  1
00030A          K CLSDTE  2
00040A*
00050A*
00060A          R JOBHST                    PFILE(JOBHSTP)
00070A          K EMPNBR  1
00080A          K *NONE   2
00090A          K JOBDTE  3
     A
```

Figure 8. Specifying the key field

Record format	Key positions		
	1	2	3
CLSHST	EMPNBR	CLSDTE	*NONE
JOBHST	EMPNBR	*NONE	JOBDTE

As in the example in the Using *NONE in the key field when creating a DDS file topic, all records from the two physical files are first merged and sequenced together on employee number (EMPNBR). However, the records for each student are merged and sequenced first on class date (CLSDTE) and then on job assignment date (JOBDTE). The set of records used for that example are now retrieved as follows:

EMPNBR	CLSDTE	JOBDTE	Description
1005	3/79		Completed class
1005	4/79		Left to begin new job
1005	6/79		Completed class
1005		4/79	Completed job
1006	3/79		Completed class
1006	5/79		Transferred to new location
1006		1/79	Completed job
1006		2/79	Completed job
1007	8/79		Left because of illness
1007		1/79	Completed job
1007		4/79	Completed job
1007		7/79	Completed job

When several adjacent record formats have *NONE in the same key position, they form a set, relative to record formats specified before and after them, that functions in sequencing as an individual record format. Key fields specified after *NONE serve to merge and sequence records of the formats within the set.

Related concepts
Using *NONE in the key field when creating a DDS file

Key fields having the same key position should not be compared under two conditions.

Example 3: Specifying the key field
The example shows how several record formats function as a set. In this example, consider a logical employee file over five physical files.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R EMPMST                    PFILE(EMPMSTP)
00020A          K EMPNBR  1
00030A*
00040A          R CLSREG                    PFILE(CLSREGP)
00050A          K EMPNBR  1
00060A          K CLSDTE  2
00070A*
00080A          R CLSHST                    PFILE(CLSHSTP)
00090A          K EMPNBR  1
00100A          K CLSDTE  2
00110A*
00120A          R JOBHST                    PFILE(JOBHSTP)
00130A          K EMPNBR  1
00140A          K *NONE   2
00150A          K JOBDTE  3
00160A*
00170A          R ACTHST                    PFILE(ACTHSTP)
00180A          K EMPNBR  1
00190A          K *NONE   2
00200A          K ACTDTE  3
     A
```

Figure 9. Specifying the key field

Record format	**Key positions**		
	1	2	3
EMPMST	EMPNBR	*NONE	*NONE
CLSREG	EMPNBR	CLSDTE	*NONE
CLSHST	EMPNBR	CLSDTE	*NONE
JOBHST	EMPNBR	*NONE	JOBDTE
ACTHST	EMPNBR	*NONE	ACTDTE

The records are merged and sequenced as follows:

1. All records are merged and sequenced by employee number.
2. For a given employee, records are sequenced by:
   a. The master record (of the EMPMST format)
   b. Records of the CLSREG and CLSHST formats, merged and sequenced together on values of CLSDTE (key position 2)
   c. Records of the JOBHST and ACTHST formats, merged together and sequenced together on values of JOBDTE and ACTDTE (key position 3)

Specifying *NONE in the key definitions achieves this sequencing as follows:

- *NONE and a field name, CLSDTE, appear in the second key position of the adjacent formats, CLSHST and JOBHST. This effectively causes a split between the two formats after the preceding key position (position 1). Records of formats above the split are merged and sequenced with records of formats below the split only on values of EMPNBR.
- An implicit *NONE in the second key position of the format EMPMST forces a similar split.
- With *NONE in key position 2, the JOBHST and ACTHST formats form a set in which the values of JOBDTE and ACTDTE are compared in order to merge and sequence records of these two formats only.

The record sequence defined by the previous key specifications is totally dependent on the order in which the formats are specified. For example, if JOBHST is specified before CLSHST, key position 2 will read:

```
*NONE, CLSDTE, *NONE, CLSDTE, *NONE
```

Here, the values of CLSDTE within CLSREG will not be sequenced with the values of CLSDTE within CLSHST, and JOBDTE will not be sequenced with ACTDTE.

Example 4: Specifying the key field
The example shows how several record formats function as a set. In this example, assume that an employee has repeated a class. To sequence two records with the same values for EMPNBR and CLSDTE, a third key field, DATE, is specified in record format CLSHST.

However, DATE cannot be specified in the next available key position (position 3) because JOBDTE and ACTDTE appear in that position for other formats. If DATE is specified in this position, the attributes of DATE are compared with the attributes of CLSHST and JOBHST, and the key definitions are rejected.

To obtain the sequencing necessary, specify *NONE before DATE, displacing DATE to key position 4.

The DATE field can be shown in position 4 as in Figure 10 on page 17.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R EMPMST                    PFILE(EMPMSTP)
00020A          K EMPNBR
00030A*
00040A          R CLSREG                    PFILE(CLSREGP)
00050A          K EMPNBR
00060A          K CLSDTE
00070A*
00080A          R CLSHST                    PFILE(CLSHSTP)
00090A          K EMPNBR
00100A          K CLSDTE
00110A          K *NONE  1
00120A          K DATE   1
00130A*
00140A          R JOBHST                    PFILE(JOBHSTP)
00150A          K EMPNBR
00160A          K *NONE
00170A          K JOBDTE
00180A*
00190A          R ACTHST                    PFILE(ACTHSTP)
00200A          K EMPNBR
00210A          K *NONE
00220A          K ACTDTE
     A
```

Figure 10. Specifying the key field

Record format	**Key positions**			
	1	2	3	4
EMPMST	EMPNBR	*NONE	*NONE	*NONE
CLSREG	EMPNBR	CLSDTE	*NONE	*NONE
CLSHST	EMPNBR	CLSDTE	*NONE	DATE
JOBHST	EMPNBR	*NONE	JOBDTE	*NONE
ACTHST	EMPNBR	*NONE	ACTDTE	*NONE

Specifying DATE in key position 4 enables records from physical file CLSHSTP with identical values for EMPNBR and CLSDTE to be merged and sequenced according to the value for DATE.

Note: Because values are actually placed in the keys to ensure the sequencing in the previous examples, duplicate key values are not always predictable when *NONE is needed for logical files with more than one record format.

Select/omit field name

You use select/omit fields to tell the operating system how to select or omit records when your program retrieves them using this record format. The only records affected are those from the physical files specified for the PFILE or JFILE keyword for this record format.

The following rules apply to select/omit fields in logical files:

- You can specify select/omit fields only if you also specify key fields or if you also specify the DYNSLT keyword for the file. You can also specify *NONE as a key field to satisfy the requirement for a key field when your application requires no key fields.
- For simple and multiple-format logical files, the operating system uses the following search order to match select/omit field names with defined fields:
  - Fields specified in DDS positions 19 through 28
  - Fields specified as parameters on the CONCAT or RENAME keyword

  If the field name is specified more than once, the first occurrence is used.

  The field name on a CONCAT or RENAME keyword and the associated field name in positions 19 through 28 cannot both be specified as select/omit fields.

  The parameter name on the SST keyword is not valid as a select/omit field unless it is defined elsewhere in the logical file record format.

  For join logical files, the select/omit field name you specify must be specified at the field level in positions 19 through 28.

When using the select/omit fields, specify either S or O in position 17. By specifying either S or O, the select and omit comparison statements are joined by OR. The system treats the select and omit comparison statements that are joined by OR independently from one another. That is, if the select or omit comparison condition is met, the record is either selected or omitted. If the condition is not met, the system proceeds to the next comparison.

By specifying a blank in position 17, the select and omit comparison statements are joined by AND. The combined comparisons must be met before the record is selected or omitted. See Figure 11 on page 19 and Figure 12 on page 19. In positions 19 through 28, specify a field name whose contents at processing time determine whether the record is to be selected or omitted based on the select/omit keyword specified for this field. The select/omit keywords are COMP, RANGE, and VALUES. The last select/omit specification can be made with the ALL keyword, but a field name is not permitted.

The field must appear in both the physical file record format and the logical file record format. Select/omit statements must follow all field and key field level entries for the record format. You can specify both select and omit for the same record format. The following information applies:

- If you specify both select and omit for a record format, the order in which you specify them is important.

  The select/omit statements are processed in the order they are specified; if a record satisfies a statement, the record is either selected or omitted as specified, and remaining select/omit statements are not examined for that record. See Figure 13 on page 19.
- If you specify both select and omit statements, you can indicate whether records not meeting any of the values specified are to be selected or omitted.
- If you do not specify the ALL keyword, the action taken for the records that do not meet the values is the converse of the type of the last statement specified. Records that do not meet selection values are omitted, and records that do not meet omission values are selected.

There are limits to the number of select/omit statements you can specify in a single logical file. If you specify many select/omit statements and you cannot create the file, reduce the overhead for the file through the following changes in the specifications, in decreasing order of importance:

- Reduce the number of record formats in the file.
- Reduce the number of physical files specified on the PFILE or JFILE keyword.
- Reduce the number of fields used (single occurrences) in the select/omit specifications.

You cannot specify a floating-point field as a select/omit field.

It is possible to have an access path with select/omit and process the file in arrival sequence. For example, CPYF can be specified with FROMRCD(1) or the high-level language cannot request keyed processing. In this case, the processing is the same as if the DYNSLT keyword had been specified.

Figure 11 on page 19 shows how to specify the select/omit field using select statements that are joined by AND.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PARTS)
00020A            PNO
00030A            DSC
00040A            UPR
00050A            QOH
00060A          K PNO
00070A          S UPR                       COMP(GT 5.00)
00080A            QOH                       COMP(LT 10)
00090A          O                           ALL
     A
```

Figure 11. Specifying the Select/Omit field (example 1)

In Figure 11 on page 19, records are selected only if they satisfy two select statements. The first statement selects records in which the value of field UPR is greater than 5.00. The second statement selects records in which the value of field QOH is less than 10. S is not specified in position 17 for field QOH. Therefore, these select statements are joined by AND. For a record to be read by a program, both conditions specified must be true.

Figure 12 on page 19 shows how to specify the select/omit field with an omit statement that is joined by OR and two select statements that are joined by AND.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PARTS)
00020A            PNO
00030A            DSC
00040A            UPR
00050A            QOH
00060A          K PNO
00070A          O DSC                       COMP(EQ 'HAMMER')
00080A          S UPR                       COMP(GT 5.00)
00090A            QOH                       COMP(LT 10)
00100A          O                           ALL
     A
```

Figure 12. Specifying the Select/Omit field (example 2)

In Figure 12 on page 19, records are supplied to the program if they pass both of the following tests:

- The DSC field is not equal to HAMMER.
- The UPR field is greater than 5.00 and the QOH field is less than 10.

Figure 13 on page 19 shows several ways to specify the same select/omit logic.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          S ST                        COMP(EQ 'NY')
00020A            REP                       COMP(EQ 'JSMITH')  1
00030A            YEAR                      COMP(LT 78)
00040A          O                           ALL
     A
00050A          O YEAR                      COMP(GE 78)
00060A          S ST                        COMP(EQ 'NY')      2
00070A            REP                       COMP(EQ 'JSMITH')
00080A          O                           ALL
     A
00090A          O REP                       COMP(NE 'JSMITH')
00100A          O ST                        COMP(NE 'NY')      3
00110A          S YEAR                      COMP(LT 78)
00120A          O                           ALL
     A
```

Figure 13. Specifying the Select/Omit field (example 3)

In Figure 13 on page 19, you want to select all the records before 1978 for a marketing representative named JSMITH in the state of New York. There are three ways to code this example:

1. All records must be compared with the select fields ST, REP, and YEAR before they can be selected or omitted.
2. All records in and after 1978 are omitted in the first comparison. Then, only the records before 1978 are compared with ST and REP. Only two select fields must be satisfied. This way is more efficient than method 1.
3. All records that are not associated with JSMITH in the state of New York are omitted in the first and second comparisons. Then, all records left are compared to YEAR. This is more efficient than method 1 or method 2.

Related reference

ALL (All) keyword—logical files only
You can use this select/omit field-level keyword to specify the action to be taken after all other select/omit specifications have been processed for this logical file.

Reference for physical and logical files (position 29)

You use this position to specify reference for physical files only.

For a logical file, leave this position blank. All logical files automatically provide the reference capability for all specified fields. Any attributes that are not specified explicitly in the logical file are furnished from the corresponding field in the physical file record format.

For a physical file, specify R in this position to refer to the attributes of a previously defined named field (called the *referenced field*). You must specify the REF or the REFFLD keyword. The referenced field can be previously defined in either the physical file you are defining or a previously created database file. The field attributes referred to are the length, data type, and decimal positions of the field, as well as the ALIAS, COLHDG, DATFMT, DATSEP, FLTPCN, REFSHIFT, TEXT, TIMFMT, TIMSEP, VARLEN, editing, and validity checking keywords.

If R is not specified, you must specify the field attributes for this field.

Note: If the DATFMT keyword is overridden on a reference field to *ISO, *EUR, *USA, or *JIS, the DATSEP keyword is not referenced.

Position 29 must be blank at the file and record levels.

The referenced field name cannot be the same as the field you are defining if that field is in the file you are defining. If the names are the same, specify the name of the file defining the referenced field as a parameter value with the REF or REFFLD keyword. If the names are different, specify the name of the referenced field with the REFFLD keyword.

To override specific attributes of the referenced field, specify those attributes for the field you are defining. In addition:

- If you specify Edit Code (EDTCDE) or Edit Word (EDTWRD) on the field, no editing specifications are copied from the referenced field.
- If you specify CHECK (AB, ME, MF, M10, M10F, M11, M11F, VN, or VNE), CHKMSGID, COMP, RANGE, or VALUES on the field, no validity checking specifications are copied from the referenced field.
- If you specify data type, field length, or decimal positions for the field you are defining, then neither editing nor validity checking keywords are copied from the referenced field.

Note: After the physical file is created, the referenced file can be deleted or changed without affecting the field descriptions in the physical file. To incorporate changes made in the referenced file, delete and re-create the physical file.

Related reference

REF (Reference) keyword—physical files only

You can use this file-level keyword to specify the name of the file from which field descriptions are retrieved.

REFFLD (Referenced Field) keyword—physical files only
You can use this field-level keyword to refer to a field under any of these conditions.

Related information
When to specify REF and REFFLD keywords for DDS files

Length for physical and logical files (positions 30 through 34)

You use these positions to specify the length of a physical or logical file field.

For a physical file, use these positions to specify the field length for each named field (unless you copy it from a referenced field). Specify the number of digits for a numeric type field, or specify the number of characters for a character type field.

For a logical file, use these positions to specify the length of a logical field. Specify the length only to override or change the length of the corresponding field in the physical file on which this logical file is based. If you leave this position blank, the field you are defining has the same length as the corresponding field in the physical file(s) on which the logical file(s) is based. If the field in the physical file is variable length and you leave the length blank, the field is also variable length in the logical file. If you do specify a length, the field in the logical file is fixed length unless you also specify the VARLEN keyword. Additionally, the SST (Substring) keyword can be used to control the length of a logical file field by specifying a character string that is a subset of another field.

If you specify length, it must be right-aligned; leading zeros are optional.

Figure 14 on page 21 shows correct and incorrect field length specifications for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5
00010A            FIELD1       7
     A
00020A            FIELD2         7
     A
00030A            FIELD3    R   +7
     A
```

Note: FIELD1 shows the field length specified incorrectly. FIELD2 and FIELD3 show the field length specified correctly.

Figure 14. Correct and incorrect length specifications for physical files

Valid length specifications are:

Data type	Valid lengths
Character	1 through 32 766 characters
Hexadecimal	1 through 32 766 bytes
Binary	1 through 18 digits
Binary character	1 through 32 766 characters
Zoned decimal	1 through 63 digits
Packed decimal	1 through 63 digits
Floating-point (single precision)	1 through 9 digits
Floating-point (double precision)	1 through 17 digits
Date	6, 8, or 10 characters
Time	8 characters
Timestamp	26 characters

The length for fields with data type L (date), T (time), or Z (timestamp) is determined by the system. You should not enter a field length in positions 30 through 34.

The field length for date and time includes the separator.

A timestamp has a fixed format that has the following form:

```
YYYY-MM-DD-hh.mm.ss.uuuuuu
```

Type in a maximum of 9 digits for single precision and 17 digits for double precision. The IBM i program supports a floating-point accuracy of 7 digits for single precision and 15 digits for double precision.

The total number of bytes occupied by all the fields in a record must not exceed 32 766 (in storage). See Table 1 on page 22 for rules on determining the total length of the record format.

The system determines the number of bytes actually occupied in storage as follows:

Data type	**Bytes occupied in storage**
Character	Number of characters
Hexadecimal	Number of bytes
Binary	
1 through 4 digits	2 bytes
5 through 9 digits	4 bytes
10 through 18 digits	8 bytes
Binary character	Number of characters
Zoned decimal	Number of digits
Packed decimal	(Number of digits/2) + 1 (truncated if fractional)
Floating-point (single precision)	4 bytes
Floating-point (double precision)	8 bytes
Date	10 characters without DATFMT keyword
	6, 8 or 10 characters with DATFMT keyword
Time	8 characters
Timestamp	26 characters

Note: The system performs arithmetic operations more efficiently for a packed decimal than for a zoned decimal data type.

Table 1 on page 22 describes the rules for determining total format length.

Table 1. Rules for determining total format length

Situation	**Action**
Does the record format contain any variable-length fields?	1. Add an extra 24 bytes to the total format length. 2. Add an extra 2 bytes to the format length for each field that is variable length.
Does the record format contain any fields that allow the null value?	Divide the total number of fields in the format by 8, round up to the next highest whole byte, then add to format length.

To override the length of a referenced field (R in position 29) in a physical file or the length of the field in a logical file, specify either a new value or a change in length. To increase the length, specify *+n*, where n is the amount of increase. To decrease the length, specify *-n*, where n is the amount of decrease. For example, type +4 to indicate that a numeric field is to be 4 digits longer than the referenced field. See

"Positional entries for physical and logical files (positions 1 through 44)" on page 4 for the example that shows how to override the field length for a physical file.

If the corresponding field in the physical file record format has a data type of binary with decimal positions greater than zero, the length cannot be overridden in the logical file. If the field you are describing is a concatenation of fields from the associated physical record format, you cannot specify the length in the logical file. The sum of the physical field lengths is calculated by the system.

If you specify a value in positions 30 through 34, your program sees the specified length. However, the length of the field in the corresponding physical file field does not change. This can cause data conversion errors. When attempting to add a member to a file or to open a member of a file, the operating system might send a mapping error message. The operating system might also send a mapping error message to your program in the following cases:

- When reading from a logical file that reduces the length specified in the corresponding physical file
- When writing to a logical file that increases the length specified in the corresponding physical file

For example, if the physical file field is defined as 4 characters long and the logical file field decreases the length to 2 characters, a value of ABCD in the physical file cannot be read by the program, although a value of AB can. In this case, the program can always write successfully. For character fields, the data is left-aligned and filled with blanks in the physical file field. For numeric fields, the data is right-aligned and filled with zeros in the physical file field.

Positions 30 through 34 are valid only for field specifications. You must leave these positions blank at the key field, select/omit field, join, record, and file level.

Note: High-level languages can impose restrictions on the field length. Any length restrictions should be observed for files used by these high-level languages.

Related concepts

Specifying record formats in a logical file in DDS
If there is more than one record format specified in a logical file, you must specify the PFILE keyword for each record format.

Related tasks

Simple- and multiple-format logical files in DDS
You must specify the PFILE keyword at the record level for simple- and multiple-format logical files. In a multiple-format logical file, a record format can use only the fields common to all the physical files specified on the PFILE keyword for that record format.

Related reference

SST (Substring) keyword—logical files only
You can use this field-level keyword to specify a character string that is a subset of an existing character, hexadecimal, zoned field, or graphic.

JFLD (Joined Fields) keyword—join logical files only
You can use this join-level keyword to identify the *from* and *to* fields whose values are used to join physical files in a join logical file. These fields are both referred to as join fields.

Data type for physical and logical files (position 35)

For a physical file, you use this position to specify the data type of the field within the database. You specify data type in a logical file only to override or change the data type of the corresponding field in the physical file on which this logical file is based.

If you leave this position blank, the field you are defining has the same data type as the corresponding field in the physical files on which the logical files are based.

Valid data type entries are as follows:

Entry
Meaning

P
Packed decimal

S
Zoned decimal

B
Binary

F
Floating-point

A
Character

H
Hexadecimal

L
Date

T
Time

Z
Timestamp

5
Binary character

Note: The data types J (only), E (either), O (open), and G (graphic) support DDS database files that use DBCS. The G (graphic) data type also supports DDS database files that use UCS-2 or UTF-16. The A (character) data type also supports database files that use UTF-8.

For physical files, if you do not specify a data type or duplicate one from a referenced field, the operating system assigns the following defaults:

- A (character) if the decimal positions 36 through 37 are blank.
- P (packed decimal) if the decimal positions 36 through 37 contain a number in the range 0 through 63.

Notes:

1. Specify 0 in position 37 to indicate an integer numeric field for packed decimal, zoned decimal, or binary fields.
2. Specify an F in position 35 for a single precision floating-point field. Use the FLTPCN keyword to specify double precision or to change the precision of an already specified floating-point field.
3. Specify an H (hexadecimal) in position 35 to indicate a field whose contents are not interpreted by the system. In most cases, hexadecimal fields are treated as character fields, except that the contents of a hexadecimal field are not translated to any character set or code page.

The following table shows what types of data conversion are valid between the data types of physical and logical file fields, where valid conversions are marked with an X or with a reference to the table notes:

Data type of physical file field	**Data type of logical file field**																
	A	**H**	**S**	**P**	**B**	**F**	**L**	**T**	**Z**	**UTF8**	**UTF16**	**UCS2**	**O**	**J**	**E**	**G**	**5**
Character (**A**)	X	X	1							X	X	X	X		X		X
Hexadecimal (**H**)	X	X	1										X	X	X		X
Zoned (**S**)	1	1	X	X	2	X											1
Packed (**P**)			X	X	2	X											
Binary (**B**)			2	2	3	2											
Floating Point (**F**)			X	X	2	X											

Data type of physical file field	Data type of logical file field																
	A	**H**	**S**	**P**	**B**	**F**	**L**	**T**	**Z**	**UTF8**	**UTF16**	**UCS2**	**O**	**J**	**E**	**G**	**5**
Date (**L**)	6, 7		6	6			X										
Time (**T**)			4					X									
Timestamp (**Z**)							5	5	X								
UTF-8	X									X	X	X	X			X	
UTF-16	X									X	X	X	X			X	
UCS-2	X									X	X	X	X			X	
Open (**O**)		X								X	X	X	X				X
Only (**J**)		X											X	X	X	X	X
Either (**E**)		X											X		X		X
Graphic (**G**)										X	X	X	X	X	X	X	
Binary character	X	X	1										X	X	X		X

Notes:

1. Valid only if the number of characters (or bytes) equals the number of digits and the character (or hexadecimal) field is not defined as a variable-length field.
2. Valid only if the binary field has a decimal precision of zero.
3. Valid only if both fields have the same decimal precision.
4. The system generates the field length for you so do not enter a length in columns 30 through 34. The length does not include the separator character.
5. Valid only if the field is input only.
6. You can specify a field length (columns 30 to 34) for these data types on a logical file field. If you do not specify a length, the system generates a default length. Valid lengths for these data types are documented with the DATFMT keyword.
7. DBCS field types are not allowed to be mapped over DATE fields.

Related tasks

Simple- and multiple-format logical files in DDS
You must specify the PFILE keyword at the record level for simple- and multiple-format logical files. In a multiple-format logical file, a record format can use only the fields common to all the physical files specified on the PFILE keyword for that record format.

Related reference

Data type (position 35)
You can use any of the four DBCS data types: J (Only), E (Either), O (Open) and G (Graphic).

Unicode considerations for database files: Data type (position 35)
The valid data types for Unicode data are the G (Graphic) data type and the A (Character) type.

JFLD (Joined Fields) keyword—join logical files only

You can use this join-level keyword to identify the *from* and *to* fields whose values are used to join physical files in a join logical file. These fields are both referred to as join fields.

Conversion of one numeric data type to another in a DDS file

Any conversion of data types from the physical file record format is permitted within the numeric types. For example, a binary field in the physical file can be converted to zoned decimal in the logical file.

Conversion between zoned decimal and character or hexadecimal in a DDS file

You can convert zoned decimal fields to character or hexadecimal fields and the converse, provided that the field lengths are the same.

The data type of the field in your program is the data type specified in the logical file. No error occurs in an I/O operation if the data passed contains only numeric characters (0 through 9). However, your program cannot send an I/O operation that attempts to pass characters other than 0 through 9 from a character or hexadecimal field to a zoned decimal field. The operating system sends a message and the I/O operation cannot be completed.

For example, suppose that a field is zoned decimal in the physical file. If you specify character type (A) for presentation to your programs, you must ensure that the field contains only numeric characters (0 through 9) when it is returned through the logical file to the physical file.

In another example, suppose a field is a character field in the physical file. If you specify the field as a zoned decimal field and as a key field in the logical file, you cannot create the logical file unless all records in the physical file contain only numeric characters (0 through 9).

Conversion of a field from floating point to packed decimal, zoned decimal, or binary in DDS

If you are converting a floating-point field (in a physical file) to a packed decimal, zoned decimal, or binary field (in a logical file), you must explicitly specify the length and decimal positions. When converting floating-point data to fixed-point format, make sure that the values you specify for length and decimal positions are large enough to accommodate the data.

Physical file length and decimal positions are presentation values only and do not indicate the magnitude of the number.

Conversion of data types when concatenating fields in DDS

If the field you are defining is a concatenation of fields from the associated physical file (specified by the CONCAT keyword), you cannot specify the data type. The operating system assigns the data type based on the data types of the fields that are being concatenated.

The general rules are:

- If the concatenation contains one or more hexadecimal (H) fields, the resulting data type is hexadecimal (H).
- If the concatenation contains one or more character (A) fields, but no hexadecimal fields, the resulting data type is character (A).
- If the concatenation contains only numeric (S, P, B) fields, the resulting data type is zoned decimal (S).
- If the concatenation contains UTF-8 fields, the result is UTF-8.
- If the concatenation contains UCS-2 or UTF-16 field, the result is UTF-16 if there is a UTF-16 field in the list; otherwise, the result is UCS-2.
- If the concatenation contains binary character fields, the result is binary character.

Conversion of data types when substringing fields in DDS

If the field you are defining is a substring of a field (specified by the SST keyword) from the logical file or the associated physical file, the original field must be character, hexadecimal, zoned, DBCS-graphic or binary character (A, H, S, or G).

If you do not specify the data type of the logical file field, then the conversion is shown in the following table (*Source field type* is the type of the physical file field or the logical file field defined earlier in the logical file source):

Source field type	Logical file field becomes:
A	A
H	H
S	A
G	G
Binary character	Binary character

Decimal positions for physical and logical files (positions 36 and 37)

You use these positions to specify the decimal placement within a packed decimal, zoned decimal, binary, or floating-point field.

If you specify the CONCAT keyword for the field you are defining, you cannot specify decimal positions. A field in the physical file that contains decimal positions cannot be included in a concatenated field.

Note: High-level languages can impose specific length and value restrictions on the decimal positions. Observe these restrictions for files used by those high-level languages.

Using decimal position with physical files

For a physical file, use these positions to specify the decimal placement within a packed decimal, zoned decimal, binary, or floating-point field. Specify a decimal number from 0 through 63 for the number of decimal positions to the right of the decimal point. (The number must not be greater than the number of digits specified in the field length.) The Positional entries for physical and logical files (positions 1 through 44) topic shows how to code the decimal positions field. If the field length is greater than 9 for a binary field, the decimal positions value must be 0.

The data is actually stored in the system without a decimal point. The decimal point is only implied. For example, the value stored for 1.23 is 123. This is what appears in display or printer files if editing is not specified.

To override the position of a referenced field (R in position 29), specify either a new value or a change in position. To increase the position, specify *+n*, where n is the amount of increase. To decrease the position, specify *-n*, where *n* is the amount of decrease. For example, an entry of +4 indicates that there are 4 more digits to the right of the decimal point than were in the referenced field. An error message is sent if the number of decimal positions is greater than the maximum allowed.

Using decimal position with logical files

For logical files, specify decimal positions only to override or change the decimal positions of the corresponding field in the physical file on which this logical file is based. If you leave these positions blank, the field you are defining has the same decimal positions as the corresponding field in the physical file on which this logical file is based.

To override or change the placement of the decimal point within a packed decimal or zoned decimal field, specify a number from 0 through 63 to indicate the number of decimal positions to the right of the decimal point. The number here must not be greater than the number of digits specified in the field length. You cannot override or change decimals when the corresponding field in the physical file is binary (data type B) and contains decimal positions greater than zero. When the logical file field is binary and

the corresponding field in the physical file is not binary (B specified in position 35 in the logical file), the decimal positions must be zero for the binary field.

You can override the position of the field by specifying a new value or by specifying an increase or decrease in position. To increase the position, specify *+n*, where n is the amount of increase. To decrease the position, specify *-n*, where n is the amount of decrease. For example, an entry of +4 indicates that there are 4 more digits to the right of the decimal point than were in the referenced field.

If you specify a value in positions 36 through 37 and your program writes or retrieves data through the logical file field to the physical file field, the operating system aligns the data on the decimal point. Depending on the case, this can cause the decimal values to be truncated, or it can cause a data conversion error. Decimal values are truncated in the following cases:

- When reading from a logical file that reduces the number of decimal positions specified in the physical file
- When writing to a logical file that increases the number of decimal positions specified in the physical file

For example, if the physical file field is defined as 4 digits long with 2 decimal positions, and the logical file field decreases the decimal positions to 0 decimal positions, a value of 0.20 in the physical file becomes a value of 0 in the logical file, and a value of 2.52 in the physical file becomes a value of 2 in the logical file.

When decimal values are truncated, the left side of the field is filled with zeros.

A data conversion error can occur in the following cases:

- When writing to a logical file that reduces the number of decimal positions specified in the physical file
- When reading from a logical file that increases the number of decimal positions specified in the physical file

The data conversion error occurs because too many digits might be moved into the space available to the left of the decimal point. For example, if, as in the previous example, the physical file field is defined as 4 digits long with 2 decimal positions and the logical file field decreases the decimal positions to 0 decimal positions, a value of 3322 written to the logical file cannot fit in the physical file. This value does not fit because only 2 digits are allowed left of the decimal point in the physical file.

To avoid data conversion errors, increase or decrease the length (positions 30 through 34) of the logical file field by the same amount that you increase or decrease the decimal positions.

Related reference

JFLD (Joined Fields) keyword—join logical files only

You can use this join-level keyword to identify the *from* and *to* fields whose values are used to join physical files in a join logical file. These fields are both referred to as join fields.

Usage for physical and logical files (position 38)

You use this field to specify that a named field is to be an input-only, both (both input and output are allowed), or neither (neither input nor output is allowed) field.

For physical files, you can specify the following entries:

Entry
Meaning

Blank
Defaults to B (both input and output allowed)

B
Both input and output allowed

Because the default is the same as the only value, you do not need to make an entry in this field.

Entries in position 38 are not referred to by the REF or REFFLD keywords. Therefore, a B in position 38 for a field in a physical file has no effect when that field is referred to in a display file.

The valid entries for logical files are described as follows:

Blank (Default)
If position 38 is blank, the following situation occurs:

- For simple and multiple format logical files (PFILE specified at the record level), the field is a both (B) field.
- For join logical files (JFILE specified at the record level), the field is an input-only (I) field.

B (Both)
If position 38 is B, the field is a *both* field and can be used for both input and output operations. That is, your program can read data from the field and write data to the field. Both fields are not valid for join logical files, because join logical files are read-only files.

I (Input-Only)
If position 38 is I, the field is an *input-only* field and can be used for input operations only. That is, your program can read data from the field, but cannot change the field. Typical cases of input-only fields are key fields (to reduce maintenance of access paths), sensitive fields that a user can see but not change (such as, in employee records, salary), and fields for which the SST or TRNTBL keyword is specified.

If your program performs a change to a record format in which you have specified input-only fields, the input-only fields are not updated and no message is sent. If your program performs an output operation to a record format in which you have specified input-only fields, the input-only fields take default values (see the DFT (Default) keyword-physical files only topic).

Input-only fields are not valid in physical files.

N (Neither)
If position 38 is N, the field is a *neither* field (neither input nor output) and is valid only for join logical files. A neither field can be used as a join field in a join logical file, but your program cannot read a neither field.

Use neither fields when the attributes of join fields in the physical files do not match. In this case, one or both join fields must be redefined. However, you might not want to include the redefined fields in the record format (that is, you might not want the application program to see the redefined fields). Therefore, code the redefined join fields as N and they do not appear in the record format.

A field with N in position 38 does not appear in the buffer used by your program. However, the field description is displayed with the Display File Field Description (DSPFFD) command.

Neither fields cannot be used as select/omit or key fields.

Entries in position 38 are not referred to using the REF or REFFLD keyword. Therefore, a B or an I in position 38 for a field in a logical file has no effect when that field is referred to in a display file.

Related reference
DFT (Default) keyword—physical files only
You can use this field-level keyword to specify a default value for a field.

JFLD (Joined Fields) keyword—join logical files only

You can use this join-level keyword to identify the *from* and *to* fields whose values are used to join physical files in a join logical file. These fields are both referred to as join fields.

Location for physical and logical files (positions 39 through 44)

These positions do not apply to physical or logical files. Leave these positions blank unless you use them for comment text.

Keyword entries for physical and logical files (positions 45 through 80)

Keyword entries are typed in positions 45 through 80 (functions).

Most of the keywords are valid for both physical and logical files. Some, however, are valid only for physical files, and some are valid only for logical files. When this is the case, the restriction is noted both in the keyword title and the text.

Specific restrictions

The following keywords are valid only for simple and multiple format logical files:

- PFILE
- REFACCPTH

The following keywords are valid only for join logical files:

- JDFTVAL
- JDUPSEQ
- JFILE
- JFLD
- JOIN
- JREF

When you use DDS to describe a source file (typically created without DDS, using the CRTSRCPF command) or when a logical file is based on a physical file to be used as a source file, you cannot use the following keywords:

ABSVAL
ALTSEQ
DESCEND
FCFO
FIFO
LIFO
NOALTSEQ
SIGNED
UNIQUE
VARLEN
ZONE

Related reference

Keyword considerations for database files that use DBCS
You should not specify DDS keywords to be used with numeric data for fields containing double-byte character set (DBCS) data. The system treats DBCS data the same as character data, and, therefore, cannot perform arithmetic operations on it.

Related information

Rules for DDS keywords and parameter values

ABSVAL (Absolute Value) keyword for physical and logical files

You can use this key-field level keyword to direct the operating system to ignore the sign of the field when the system sequences the values associated with this numeric field.

This keyword has no parameters.

The following example shows six records with a zoned decimal key field:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
1	98	F9F8
2	00	F0F0
3	98-	F9D8
4	97	F9F7
5	20	F2F0
6	99	F9F9

If you do not specify any sequencing keywords or the ALTSEQ keyword, the default sequencing for the key field is the SIGNED attribute. In this case, the records are sequenced in the following order:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
3	98-	F9D8
2	00	F0F0
5	20	F2F0
4	97	F9F7
1	98	F9F8
6	99	F9F9

If you specify the ABSVAL keyword, the absolute value of the negative field is used, and the resulting sequence is:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
2	00	F0D0
5	20	F2F0
4	97	F9F7
1	98	F9F8
3	98-	F9D8
6	99	F9F9

The ABSVAL keyword is not valid for a character, date, time, timestamp, and hexadecimal data type field. You cannot use this keyword with the DIGIT, SIGNED, UNSIGNED, or ZONE keywords.

ABSVAL (a key field-level keyword) causes ALTSEQ (a file-level keyword) to be ignored. If you specify ABSVAL for a key field, NOALTSEQ is in effect for that key field, even if ALTSEQ was specified at the file level. This occurs, even if the NOALTSEQ keyword is specified.

Example

The following example shows how to specify the ABSVAL keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A            ORDAMT         5  0
00020A          K ORDAMT                    ABSVAL
     A
```

ALIAS (Alternative Name) keyword for physical and logical files

You can use this field-level keyword to specify an alternative name for a field. When the program is compiled, the alternative name is brought into the program instead of the DDS field name.

The high-level language compiler in use determines if the ALIAS name is used. See the appropriate high-level language reference manual for information about ALIAS support for that language.

The format of the keyword is:

```
ALIAS(alternative-name)
```

The alternative name must be different from all other alternative names and from all DDS field names in the record format. If a duplicate is found, an error message appears on the field name or alternative name.

An alternative name cannot be used within DDS or any other IBM i function (for example, as a key field name, as the field name specified for the REFFLD keyword, or as a field name used in the Copy File (CPYF) command).

When you refer to a field that has the ALIAS keyword, the ALIAS keyword is copied in unless the ALIAS keyword is explicitly specified on the referencing field.

Example

The following example shows how to specify the ALIAS keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A            FIELDA        25A         ALIAS(CUSTOMERNAME)
     A
```

In the example, the alternative name for FIELDA is CUSTOMERNAME.

Related information

Rules for DDS keywords and parameter values

ALL (All) keyword—logical files only

You can use this select/omit field-level keyword to specify the action to be taken after all other select/omit specifications have been processed for this logical file.

Specify ALL with S in position 17 to direct the operating system to select any records that do not meet any of the other select/omit rules. Specify O in position 17 to direct the operating system to omit any records that do not meet any of the other select/omit rules. If specified, ALL must follow the other select/omit keywords. You cannot specify a field name with the ALL keyword.

This keyword has no parameters.

If you do not specify the ALL keyword, the default action taken is the opposite of the last select/omit specification you made for the file. If the last specification was a select, the default is to omit all. If the last specification was an omit, the default is to select all.

Example

The following example shows how to specify the ALL keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          S ACT                       COMP(EQ 3000)
00020A          S ACT                       COMP(GT 3100)
00030A          O AMT                       COMP(LT 0)
00040A          O                           ALL
     A
```

Related reference

Select/omit field name

You use select/omit fields to tell the operating system how to select or omit records when your program retrieves them using this record format. The only records affected are those from the physical files specified for the PFILE or JFILE keyword for this record format.

ALTSEQ (Alternative Collating Sequence) keyword for physical and logical files

You can use this file-level keyword to direct the operating system to use an alternative collating sequence table when the system sequences the records of a file member for retrieval, if you specified a key for this file.

The format of the keyword is:

```
ALTSEQ([library-name/]table-name)
```

The name of the alternative collating sequence table is a required parameter value. The library-name is optional. If you do not specify the library-name, the IBM i operating system uses the library list (*LIBL) at file creation time.

The ALTSEQ keyword is not valid under the following conditions:

- When you specify FILETYPE(*SRC) on the Create Physical File (CRTPF) or Create Logical File (CRTLF) command.
- When key fields have a data type of packed decimal, binary, or floating-point.
- When key fields are specified with ABSVAL or SIGNED. For those fields, NOALTSEQ (a key field-level keyword) is assumed and does not need to be specified. You can specify NOALTSEQ for any field in a composite key that does not require the alternative sequence.
- When you specify a value other than *SRC on the SRTSEQ parameter on the Create Physical File (CRTPF) or Create Logical File (CRTLF) command.

The ALTSEQ keyword cannot be specified with the REFACCPTH keyword.

You must have use authority to the alternative collating sequence table. The alternative collating sequence table is created using the Create Table (CRTTBL) command.

ALTSEQ causes zoned key fields to default to unsigned sequence. You can override the default by specifying the SIGNED keyword for individual key fields.

Example

The following example shows how to specify the ALTSEQ keyword for a logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                      ALTSEQ(TABLELIB/TABLE1)
00020A          R RECORD1                   PFILE (PF1)
00030A            :
00040A            :
00050A            :
00060A            NAME          20
00070A            :
00080A            :
```

```
00090A          K NAME
     A
```

Records with format RECORD1 are sequenced by key NAME according to the alternative collating sequence table (TABLE1 in library TABLELIB).

ALWNULL (Allow Null Value) keyword—physical files only

You can use this field-level keyword to define this field to allow the null value.

This keyword has no parameters.

When you specify the ALWNULL keyword, the maximum length you can specify in positions 30 to 34 is 32 765 bytes (32 739 if the field is also variable length).

For physical files, when you specify the DATFMT keyword with values of *JOB, *MDY, *DMY, *YMD, or *JUL and the field allows null value, you must specify a valid date on the DFT keyword for this field.

Example

The following example shows how to specify the ALWNULL keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1
00020A            FIELD1        75A         ALWNULL
00030A            FIELD2       100A
00040A            FIELD3          L         ALWNULL
00050A                                      DATFMT(*MDY)
00060A                                      DFT('12/25/93')
     A
```

FIELD1 is defined to allow the null value. The default value of FIELD1 is the null value. FIELD2 is defined to not allow the null value. The default value of FIELD2 is blanks.

CCSID (Coded Character Set Identifier) keyword for physical and logical files

You can use this keyword to specify a coded character set identifier (CCSID) for character fields. The CCSID keyword is a file- or field-level keyword on physical files, and a field-level keyword on logical files.

The format of the keyword is:

```
CCSID(value [field-display-length | *MIN | *LEN display-positions]
      [*CONVERT | *NOCONVERT] [*NORMALIZE])
```

The value is a number up to 5 digits long that identifies a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other relevant information that uniquely identifies the coded graphic character representation used for the data in the field.

For logical files, the following characteristics must be true before the CCSID keyword is allowed on a logical file field.

- If the specified value on the logical file CCSID keyword uses a Unicode encoding scheme, then the field data type must be G for a UCS-2 Level 1 or a UTF-16 encoding scheme, and the field data type must be A for a UTF-8 encoding scheme. Also, the corresponding physical file field must be of types A, G, or O.
- If the specified value on the logical file CCSID keyword does not use the Unicode encoding scheme, then the field data type must be A, O, or G. Also, the corresponding physical file field must be a G type field and have the CCSID keyword specified with a UCS-2 or UTF-16 value, or be an A (character) type field with a UTF-8 CCSID.

The field-display-length parameter is optional and is only used when the field is referenced by a field in a display file. The parameter is only valid when the value parameter is UCS-2 or UTF-16. The field-display-length allows the user to control the field size according to the UCS-2 or UTF-16 data.

A special value, *MIN, can be specified instead of a field-display-length. It can be defined in a physical file only for use by a referencing field in a display file DDS record format. This value is used to specify a field

display length defined in terms of display positions. This value causes the field length on the screen to be equal to the field length defined in the DDS.

A special value, *LEN, along with the display-positions value can be specified instead of a field-display-length. It can be defined in the physical file only for use by a referencing field in a display file DDS record format. This value is used to specify a field display length defined in terms of display positions. This value causes the field length on the screen to be equal to the display-positions value.

The *CONVERT parameter is optional. It can be defined in a physical file only for use by a referencing field in a printer file DDS record format. The parameter specifies that, when the field prints, the UCS-2 or UTF-16 data is converted to the target CCSID specified on the CHRID command parameter on the CRTPRTF, CHGPRTF, or OVRPRTF command. If you do not specify this parameter, the keyword is set to *CONVERT as default. If you specify *NOCONVERT, the UCS-2 or UTF-16 data will be not converted to the target CCSID.

The *NORMALIZE parameter is optional but provides more predictable results when you are using UTF-8 and UTF-16 data. You can use this parameter to combine characters in UTF-8 and UTF-16 data. This support for combining characters allows a resulting character to be composed of more than one character. After the first character, up to 300 different nonspacing accent characters, such as umlaut and accent, can follow in the data string. If the resulting character is one that is already defined in the character set, normalization replaces the string of combining characters with the hexadecimal value of the defined character. If the resulting character is not a defined character, the combining character string is unchanged after normalization. For example, normalization of a UTF-16 graphic string of an 'e' (X'0065') followed by an acute character (X'0301') results in the replacement character é (X'00E9').

You can use the *NORMALIZE parameter only when the CCSID keyword is used at the field level. Without this keyword, the system assumes that data inserted or updated into UTF-8 and UTF-16 fields is already normalized. *NORMALIZE is valid only with a CCSID keyword UTF-16 value (on a graphic field) or UTF-8 value (on a character field).

When specified at the file level for physical files, the CCSID keyword applies to each character field in the file except those character fields that also have the CCSID keyword specified. If the file level CCSID is UCS-2 or UTF-16, it is applied to any G field that does not have a CCSID keyword. If a CCSID value on the physical file field used the UCS-2 encoding scheme, the data type of this field must be type G. If a CCSID value on the physical file field used the UTF-8 encoding scheme, the data type of this field must be character. If a CCSID value on the physical file field used the UTF-16 encoding scheme, the data type of this field must be type G.

If the CCSID keyword is not specified at the file level and not all character fields have the CCSID keyword specified, then the fields are assigned the job's default CCSID when the file is created.

Examples

The following example shows how to specify the CCSID keyword for physical files.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                     CCSID(285)
00020A          R RECORD1
00030A            FIELD1        75G       CCSID(13488)
00040A            FIELD2       150A
00050A            FIELD3        20A
00060A            FIELD4        10A       CCSID(1208 *NORMALIZE)
00070A            FIELD5        10G       CCSID(1200)
     A
```

FIELD1 is assigned a UCS-2-ccsid value of 13488. FIELD2 and FIELD3 are assigned a CCSID value of 285. FIELD4 is assigned a UTF-8 CCSID value of 1208 and its data will be normalized before being inserted or updated in the file. FIELD5 is assigned a UTF-16 CCSID value of 1200 and its data will not be normalized before being inserted or updated in the file.

The following example shows how to specify the CCSID keyword on a corresponding logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00000A
00010A          R RECORD1
```

```
00020A            FIELD1        75A         CCSID(37)
00030A            FIELD2       150G         CCSID(13488 80)
00040A            FIELD3        20A
00050A            FIELD4        10G         CCSID(1200 *NORMALIZE)
00060A            FIELD5        10A
     A
```

The logical file's FIELD1 is assigned a SBCS CCSID value of 37. Conversion occurs between the physical file and the logical file for FIELD1 because the physical file field contains UCS-2 data. The logical file's FIELD2 is assigned a UCS-2-ccsid value of 13488. Conversion occurs between the physical file and the logical file for FIELD2 because the logical file contains UCS-2 data. A CCSID is not specified for FIELD3. FIELD4 is assigned a UTF-16 CCSID value of 1200. Conversion occurs between the physical file and the logical file for FIELD4 because the physical file field contains UTF-8 character data. The data will be normalized. FIELD5 is assigned the CCSID of the job in which the file is created. Conversion occurs between the physical file and the logical file for FIELD5 because the physical file field contains UTF-16 data. The data will not be normalized.

Related information

CCSID (Coded Character Set Identifier) keyword

i5/OS globalization

CHECK (Check) keyword for physical and logical files

You can use this field-level keyword to specify validity checking in display files.

The format of the keyword is:

```
CHECK(edit-check-code [. . .])
```

The CHECK keyword does not affect the physical or logical file that is defined. When you define an input-capable field in a display file, refer to the field you are defining by specifying R in position 29 and using the REF or REFFLD keyword. At display file creation, the operating system copies the CHECK keyword and other field attributes from the field in the physical or logical file into the field in the display file. You can override the CHECK keyword (as well as all other validity checking keywords and the CHKMSGID keyword) by specifying any validity checking keyword for the field in the display file.

The rules for specifying this keyword in a physical or logical file are similar to those for a display file. However, only the following codes are allowed in physical or logical files:

Code
Meaning

AB
Allow blank

ME
Mandatory enter

MF
Mandatory fill

M10
IBM Modulus 10 self-check algorithm

M10F
IBM Modulus 10 self-check algorithm

M11
IBM Modulus 11 self-check algorithm

M11F
IBM Modulus 11 self-check algorithm

VN
Validate name

VNE
Validate name extended

You cannot specify the CHECK(AB), CHECK(VN), CHECK(VNE), CHECK(M10), CHECK(M11), CHECK(M10F), or CHECK(M11F) keywords on a floating-point field (F in position 35). You cannot specify the CHECK keyword on a hexadecimal field (H in position 35). Do not specify the CHECK keyword on a date, time, or timestamp field (L, T, or Z in position 35).

Related information

Reference for display files (position 29)
CHECK (Check) keyword for display files

CHKMSGID (Check Message Identifier) keyword for physical and logical files

You can use this field-level keyword to identify an error message that is associated with validity checking keywords.

If the CHKMSGID keyword is not specified, a system-supplied message is used. If the CHKMSGID keyword is specified and the field you are now defining is referred to later during display file creation, the validity checking information and the CHKMSGID keyword are copied into the display file. If a validity checking error is found while checking input from the screen, the error message specified on the CHKMSGID keyword is displayed on the message line.

CHKMSGID does not affect the physical or logical file you are defining.

The format of the keyword is:

```
CHKMSGID(message-id [library/]message-file [message-data-field])
```

If the message-data-field parameter is specified, the field it identifies does not need to be defined in the physical or logical file. However, if the field containing the CHKMSGID keyword is referred to during display file creation, the message data field must be defined in the display file (in the same record format as the field with the CHKMSGID keyword).

CHKMSGID is allowed only on fields that also contain a VALUES, RANGE, CMP, COMP, CHECK(M10), CHECK(M11), CHECK(VN), or CHECK(VNE) keyword.

Related information

CHKMSGID (Check Message Identifier) keyword for display files

CMP (Comparison) keyword for physical and logical files

This keyword is equivalent to the COMP keyword.

The format of the keyword is:

```
CMP(relational-operator value)
```

The COMP keyword is preferred.

Related reference

COMP (Comparison) keyword for physical and logical files
You can use this field-level keyword to specify validity checking for the field you are defining when the field is referred to later during display file creation. The COMP keyword is equivalent to the CMP keyword.

COLHDG (Column Heading) keyword for physical and logical files

You can use this field-level keyword to specify column headings used as a label for this field by text management, the query utility, the data file utility (DFU), and the screen design aid (SDA).

The format of the keyword is:

```
COLHDG('line-1' ['line-2' ['line-3']])
```

A maximum of three lines of 20 characters each is allowed. Each line of the column heading must be enclosed in single quotation marks ('). Use double single quotation marks (' ') to specify single quotation

marks within column headings. Use one or more blanks to separate the first column heading line from the second and the second from the third.

For a physical file, if you do not specify COLHDG and it is not retrieved from a referenced field, the field name is used. If you do not specify COLHDG for a logical file, the column heading from the physical file is used, except when the field is a concatenation of fields; in this case, the default is the field name.

If you specify COLHDG but do not specify TEXT, 50 positions of column heading information are used as text. For example, COLHDG('Order' 'Date') is equivalent to TEXT('Order Date').

Example

The following example shows how to specify the COLHDG keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00150A            ORDDAT         5  0       COLHDG('Order' 'Date')
00160A            NAME          20          COLHDG('Customer''s Name')
00170A            CITY          20          COLHDG('Customer' 'City' 'Field')
     A
```

Decimal positions or data type must be specified for ORDDAT because Order Date is a numeric field (denoted by NNNNN as follows).

The following display illustrates how the column headings can appear when running text management, query, DFU, or SDA.

```
Customer
Order                                  City
Date       Customer's Name             Field
NNNNN      XXXXXXXXXXXXXXXXXXX         XXXXXXXXXXXXXXXXXXX
```

COMP (Comparison) keyword for physical and logical files

You can use this field-level keyword to specify validity checking for the field you are defining when the field is referred to later during display file creation. The COMP keyword is equivalent to the CMP keyword.

For logical files, you can also specify this keyword at the select/omit-field level.

The format of the keyword is:

```
COMP(relational-operator value)
```

At the select/omit-field level, the format of the keyword is:

```
COMP(relational-operator field-name)
```

Valid relational operators are:

Relational operator
Meaning

EQ
Equal to

NE
Not equal to

LT
Less than

NL
Not less than

GT
Greater than

NG
Not greater than

LE
Less than or equal to

GE
Greater than or equal to

Specify the value parameter at either the field level or the select/omit field level. Specify the field name parameter only at the select/omit field level.

Example 1

The following example shows how to specify the COMP keyword for character and numeric strings.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD                    PFILE(PF1)
00020A
00030A            FIELDA         1  0       COMP(NE 0)     1
C0040A            FIELDB         1          COMP(NE 'A')   1
00050A            FIELDC
00060A            FIELDD
00070A            FIELDE
00080A          K FIELDB
00090A          S FIELDC                    COMP(EQ FIELDD)   2
00100A          S FIELDA                    COMP(NE 0)        2
00110A          S FIELDE                    COMP(NE *NULL)    2
00120A          O FIELDB                    COMP(GE 'A')      2
     A
```

1
COMP is specified for FIELDA and FIELDB as a validity checking keyword for display files that refer to FIELDA and FIELDB.

2
COMP is specified as a select/omit keyword for FIELDC, FIELDA, FIELDB, and FIELDE. Records from the physical file PF1 are retrieved through this logical file record format depending on the following comparisons:

- FIELDC: Records are selected when FIELDC equals FIELDD.
- FIELDA: Records not meeting FIELDC test are selected only when FIELDA is not equal to zero.
- FIELDE: Records not meeting FIELDA test are selected only when FIELDE is not the null value.

Example 2

The following example specifies the COMP keyword using a hexadecimal character string.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RCD1                      PFILE(PF1)
00020A            CODEA
00030A            FLD1
00040A            FLD2
00050A          K FLD1
00060A          S CODEA                     COMP(EQ X'51')
     A
```

COMP is specified as a select/omit keyword for CODEA (which is a 1-byte field). Records from physical file PF1 are retrieved through this record format only if the value of field CODEA is hex 51.

Related reference
CMP (Comparison) keyword for physical and logical files
This keyword is equivalent to the COMP keyword.

Specifying COMP at the field level

At the field level, the COMP keyword does not affect the physical or logical file you are describing.

However, when you describe an input-capable field in a display file, you can refer to the field you are describing by specifying R in position 29 and the REF or REFFLD keyword. During display file creation,

the operating system copies the COMP keyword and other field attributes from the field in the logical file into the field in the display file. You can override the COMP keyword (as well as all other validity checking keywords and the CHKMSGID keyword) by specifying any validity checking keyword for the field in the display file.

You cannot specify a field name as a parameter value for a field-level COMP keyword.

You cannot specify *NULL as a parameter value for a field level COMP keyword.

You cannot specify the COMP keyword on a floating-point field (F in position 35) or a hexadecimal field (H in position 35). Do not specify the COMP keyword on a date, time, or timestamp field (L, T, or Z in position 35).

The rules for specifying this keyword in a physical or logical file are the same as the rules for a display file.

Defining a numeric field for physical and logical files

When a workstation user types in data, the operating system aligns the characters typed in according to the number of decimal positions in the field. Leading and trailing blanks are filled with zeros when the field is passed to your program. If you do not type a decimal character, the operating system places a decimal character to the right of the farthest right character typed. For example, for a numeric field with a length of 5 (specified in position 34) and 2 decimal positions (specified in position 37), 1.2 is interpreted as 001.20, and 100 is interpreted as 100.00.

Related information

Reference for display files (position 29)

COMP keyword for display files

Specifying COMP at the select/omit-field level

At the select/omit-field level, you can specify a field name, a value, or *NULL for the parameter.

If the select/omit field is a binary character field, the field-name parameter must also be a binary character field. The comparisons for binary character select/omit fields also need to take the actual lengths of the operands into consideration. The operands will only compare as equal if the actual lengths of the operands are equal. Shorter operands will be considered less than the longer operands when they are equal up to the length of the shorter operand.

If you specify a value, the following rules apply:

- If you are defining a character field, specify a character constant or a hexadecimal character string.

  Specify character strings with single quotation marks. See Example 1 in COMP (Comparison) keyword for physical and logical files.

  Specify hexadecimal character strings as an X followed by a combination of the digits 0 through 9 and the letters A through F, enclosed in single quotation marks ('). The number of hexadecimal digits in single quotation marks must be exactly twice the specified length of the field. See Example 2 in COMP (Comparison) keyword for physical and logical files.
- If you are defining a numeric field, specify a numeric string (digits 0 through 9 specified without single quotation marks).
- If you are defining a date field, specify a valid date in the same format specified on the DATFMT keyword and use the same separator as specified on the DATSEP keyword.

  For example, `COMP(EQ '12/15/05')` is the default value if *MDY is specified for DATFMT and '/' is specified for DATSEP.
- If you are defining a time field, specify a valid time in the same format specified on the TIMFMT keyword and use the same separator as specified on the TIMSEP keyword.

  For example, `COMP(EQ '11.00.00')` is the default value if *ISO is specified for TIMFMT. The default separator for *ISO is a period (.).
- If you are defining a timestamp field, you must specify the default value in the following format:

```
COMP(EQ 'YYYY-MM-DD-HH.MM.SS.UUUUUU')
```

If you specify *NULL, the relational operator must be EQ or NE.

COMP selects or omits records retrieved from the physical file on which this logical file is based when your program sends an input operation to the record format you are defining. The operating system selects or omits records as a result of testing the value of the select/omit fields against the value you specify, the value of the field whose name you specify, or the null value (if *NULL was specified).

CONCAT (Concatenate) keyword—logical files only

You can use this field-level keyword to combine two or more fields from the physical-file record format into one field in the logical-file record format you are defining. The name of this concatenated field must appear in positions 19 through 28.

The format of the keyword is:

```
CONCAT(field-1 field-2...)
```

Specify the physical file field names in the order in which you want them to be concatenated, and separate them by blanks.

If the same physical field is specified more than once in a record format in the logical file (that is, by using either RENAME or CONCAT), the sequence in which the fields are specified in the logical file is the sequence in which the data is moved to the physical file on an update or insert operation. Thus, the value in the last occurrence of the physical field is the value that is put in the physical record and is the value that is used for all keys built over that physical field. All previous values of the same field are ignored.

If you want to use a field defined using the CONCAT keyword or a field specified as a parameter value on the CONCAT keyword as a key field, see the Key field name topic.

Binary character fields can be concatenated only with other binary character fields. UTF-8 fields can be concatenated only with other UTF-8 fields. UCS-2 and UTF-16 fields can be concatenated only with fields of the same type or with each other.

You cannot include a field containing decimal positions other than zero in a concatenated field. You can include a field having decimal positions of zero in which case the field is treated as an integer field.

The IBM i program assigns the length of the concatenated field as the sum of the lengths (digits and characters) of the fields included in the concatenation.

The operating system assigns the field to be fixed length or variable length based on the fields that are concatenated. The general rules are:

- Concatenation of a variable-length field to either a fixed-length field or another variable-length field results in a variable-length field.
- Concatenation of a fixed-length field to a fixed-length field results in a fixed-length field unless the VARLEN keyword is also specified on the same field as the CONCAT keyword.

  Note: If the result of the concatenation is a variable-length field, a field that allows the null value, a UCS-2 field, a UTF-16 field, a UTF-8 field, or a binary character field, the CONCAT field must be input only (I in position 38). If a logical file record format contains a concatenation, it cannot contain any fields that allow the null value from the physical file record format of the based-on file.

The operating system assigns the data type based on the data types of the fields that are being concatenated. The general rules are:

- If the concatenation contains one or more hexadecimal (H) fields, the resulting data type is hexadecimal (H).
- If the concatenation contains one or more character (A) fields, but no hexadecimal fields, the resulting data type is character (A).
- If the concatenation contains only numeric (S, P, B) fields, the resulting data type is zoned decimal (S).
- If the concatenation contains UTF-8 fields, the result is a UTF-8 field.

- If the concatenation contains UCS-2 or UTF-16 field, the result is UTF-16 if there is at least one UTF-16 field in the list; otherwise, the result is UCS-2.
- If the concatenation contains binary character fields, the result is binary character.

When concatenating numeric fields, the sign of the farthest right field in the concatenation is used as the sign of the concatenated field. The signs of the other fields are ignored; however, they are present in the concatenated field. Therefore, if a negative value appears in a field other than the last field, you must take appropriate action to delete the embedded signs (such as converting the concatenated field to packed decimal).

The maximum length of a concatenated field varies, depending on the data type of the concatenated field and the length of the fields being concatenated. If the concatenated field is zoned decimal (S), its total length cannot exceed 63 bytes. If the field is character (A) or hexadecimal (H), its total length cannot exceed 32 766 bytes. If the concatenated field is a variable length field, its total length cannot exceed 32 740 (32 739 if the field also allows the null value).

You cannot include a floating-point, date, time, or timestamp field in a concatenated field.

In join logical files, the fields to be concatenated must be from the same physical file. The first field specified on the CONCAT keyword identifies which physical file is used. The first field must, therefore, be unique among the physical files the join logical file is based on, or you must also specify the JREF keyword following the CONCAT keyword to specify which physical file to use.

Examples

The following examples show how to specify the CONCAT keyword.

Example 1

MTH, DAY, and YEAR are fields in the physical file that are concatenated into one field DATE in the logical file, as shown in the following example.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PF1)
00020A            DATE                      CONCAT(MTH DAY YEAR)
     A
```

Example 2

In the following example, if the program changes DATE from 01 03 81 to 02 05 81, the value placed in the physical record does not change because the fields specified last are MTH (value 01), DAY (value 03), and YEAR (value 81). However, if MTH, DAY, and YEAR are changed to new values, the value of DATE in the physical record also changes.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD2                   PFILE(PF1)
00020A            DATE                      CONCAT(MTH DAY YEAR)
00030A            MTH
00040A            DAY
00050A            YEAR
     A
```

Example 3

In the following example, fields from the physical file are concatenated into more than one field in the logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD3                   PFILE(PF1)
00020A            DATE                      CONCAT(MTH DAY YEAR)
00030A            CMPDAT                    CONCAT(DAY MTH YEAR)
     A
```

Example 4

In the following example, if the fields from PF1 are:

- FIXED1 is a fixed length field.
- FIXED2 is a fixed length field.
- VARLEN1 is a variable length field.

The resulting fields are:

- FIELD1 is a variable length field.
- FIELD2 is a fixed length field.
- FIELD3 is a variable length field.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD4                   PFILE(PF1)
00020A            FIELD1                    CONCAT(FIXED1 VARLEN1)
00030A            FIELD2                    CONCAT(FIXED1 FIXED2)
00040A            FIELD3                    CONCAT(FIXED1 FIXED2)
00050A                                      VARLEN
     A
```

Related concepts

Field name
When position 17 is left blank, the name specified in positions 19 through 28 is a field name.

Key field name
When you specify K in position 17, the name specified in positions 19 through 28 is a key field name.

Related reference

CONCAT (Concatenate) keyword
Using this field-level keyword, you can combine two or more fields from the physical-file record format into one field in the logical-file record format you are describing.

DATFMT (Date Format) keyword for physical and logical files

You can use this field-level keyword to specify the format of a date field. This keyword is valid only for date fields (data type L) or for logical-file zoned fields (data type S), packed fields (data type P), or character fields (data type A) whose corresponding physical file fields are date fields (data type L).

The format of the keyword is:

```
DATFMT(date-format)
```

The date-format parameter specifies the format for the date. The following table describes the valid date formats and their default separator values for physical file fields.

Format name	Date format parameter	Date format and separator	Field length	Example
Job Default	*JOB[1]			
Month/Day/Year	*MDY[1]	mm/dd/yy	8	06/21/90
Day/Month/Year	*DMY[1]	dd/mm/yy	8	21/06/90
Year/Month/Day	*YMD[1]	yy/mm/dd	8	90/06/21
Julian	*JUL[1]	yy/ddd	6	90/172
International Standards Organization	*ISO	yyyy-mm-dd	10	1990-06-21
IBM USA Standard	*USA	mm/dd/yyyy	10	06/21/1990
IBM European Standard	*EUR	dd.mm.yyyy	10	21.06.1990

Format name	Date format parameter	Date format and separator	Field length	Example
Japanese Industrial Standard Christian Era	*JIS	yyyy-mm-dd	10	1990-06-21

Note: If this format is specified and the field allows the null value, you must specify a valid date for the DFT keyword for this field.

Other attributes of the DATFMT keyword for physical file fields are:

- You can specify only the DATFMT keyword on the date (L) data type.
- If you do not specify the DATFMT keyword, the default is *ISO.
- Field length values and decimal position values must be blank.

The following table describes the valid date formats and their default separator values for logical files.

Format name	Date format parameter	Date format	Zoned or character field length	Zoned or character example	Packed field length	Packed example (in Hex)
Job default	*JOB					
Month/Day/Year	*MDY	mmddyy	6,0	062196	6,0 or 7,0	'0062196F'X
Day/Month/Year	*DMY	ddmmyy	6,0	210696	6,0 or 7,0	'0210696F'X
Year/Month/Day	*YMD	yymmdd	6,0	960621	6,0 or 7,0	'0960621F'X
Month/Day/Year (4 digit year)	*MDYY[1]	mmddyyyy	8,0	06211996	8,0 or 9,0	'006211996F'X
Day/Month/Year (4 digit year)	*DMYY[1]	ddmmyyyy	8,0	21061996	8,0 or 9,0	'021062006F'X
Year/Month/Day (digit year)	*YYMD[1]	yyyymmdd	8,0	19960621	8,0 or 9,0	'019960621F'X
Julian	*JUL	yyddd	5,0	96172	5,0	'96172F'X
Julian (4 digit year)	*LONGJUL[1]	yyyyddd	7,0	1996172	7,0	'1996172F'X
Century/Day/Month/Year	*CMDY[1]	cmmddyy	7,0	0062196	7,0	'0062196F'X
Century/Day/Month/Year	*CDMY[1]	cddmmyy	7,0	1210696	7,0	'1210696F'X
Century/Year/Month/Day	*CYMD[1]	cyymmdd	7,0	1960621	7,0	'1960621F'X
Month/Year	*MY[1,2]	mmyy	4,0	0696	4,0 or 5,0	'00696F'X
Year/Month	*YM[1,2]	yymm	4,0	9606	4,0 or 5,0	'09606F'X
Month/Year (4 digit year)	*MYY[1,2]	mmyyyy	6,0	061996	6,0 or 7,0	'0061996F'X
Year/Month (4 digit year)	*YYM[1,2]	yyyymm	6,0	199606	6,0 or 7,0	'0199606F'X
International Standards Organization	*ISO	yyyymmdd	8,0	19960621	8,0 or 9,0	'019960621F'X
IBM USA Standard	*USA	mmddyyyy	8,0	19960621	8,0 or 9,0	'006211996F'X
IBM European Standard	*EUR	ddmmyyyy	8,0	21061996	8,0 or 9,0	'021061996F'X
Japanese Industrial Standard Christian Era	*JIS	yyyymmdd	8,0	19960621	8,0 or 9,0	'019960621F'X

Notes:

1. These DATFMTs are not valid for the date (L) type field. They are only valid on logical file zoned, packed, or character types having a physical file based on date type fields.
2. DATFMTs that do not have any "days" specified are implied to be day 1 of the specified month.

Other attributes of the DATFMT keyword specified for logical file fields are:

- The packed (P), zoned (S), character (A), and date (L) data types for logical file fields allow the DATFMT keyword.
- Field length can be specified for packed, character, and zoned logical file fields, but must be a valid value listed in the table.
- If you do not specify the DATFMT keyword and the data type is L, the default is the date format and field length from the corresponding physical file field.
- For packed and zoned data types, the decimal positions (positions 36 and 37) must be blank.
- For the packed data type, two lengths are sometimes allowed for a particular format. The larger length is better from a performance perspective. If you do not specify a length, the smaller length is used as the default.

Attributes of the DATFMT keyword that apply to both physical file fields and logical file fields include the following situations:

- If you specify *JOB, the default is the job attribute and the field length and is based on the job attribute without separators.
- If the DFT keyword is not specified, the default value is the current date.
- If you specify the *ISO, *USA, *EUR, or *JIS value, you cannot specify the DATSEP keyword. These date formats have a fixed separator.
- The DATFMT keyword overrides the job attribute for a date field. It does not change the system default.

Example

The following example shows how to specify the DATFMT keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A
00020A          R RECORD
00030A            DATFLD1         L         DATFMT(*JUL)
00040A            DATFLD2         L         DATFMT(*EUR)
     A
```

If the current date is June 21, 1990, the current system date format value is MDY, and the current system separator is /, DATFLD1 contains 90/172 (the 172nd day of the year 1990). DATFLD2 contains 21.06.1990.

DATSEP (Date Separator) keyword for physical and logical files

You can use this field-level keyword to specify the separator character for a date field. This keyword is valid only for date fields (data type L).

The format of the keyword is:

```
DATSEP(*JOB | 'date-separator')
```

The date separator parameter specifies the separator character that appears between the year, month, and day. Valid values are a slash (/), dash (–), period (.), comma (,) or blank ( ). The parameter must be enclosed in single quotation marks (').

If you specify *JOB, the default is the job attribute.

For physical files, if you do not specify the DATSEP keyword, the default is the job attribute.

For logical files, if you do not specify the DATSEP keyword, the default is the date separator from the physical file. If you did not specify the DATSEP keyword for the physical file field (*ISO, *USA, *EUR, or *JIS was specified on the DATFMT keyword), the default for DATSEP is the job attribute.

If you specify the *ISO, *USA, *EUR, or *JIS date format value on the DATFMT keyword, you cannot specify the DATSEP keyword. These formats have a fixed date separator.

The DATSEP keyword overrides the job attribute. It does not change the system default.

Example

The following example shows how to specify the DATSEP keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A
00020A          R RECORD1
00030A            DATFLD2         L         DATFMT(*DMY) DATSEP('-')
00040A            DATFLD4         L         DATSEP(' ')
     A
```

If the current date is June 21, 1990, the current system date format value is MDY, and the system date separator value is '/', DATFLD2 contains 21-06-90. DATFLD4 contains 06 21 90.

DESCEND (Descend) keyword for physical and logical files

You can use this key field-level keyword to specify that the values of this character, hexadecimal, or numeric key field are retrieved in descending sequence.

The default is ascending sequence. See SIGNED (Signed) keyword for physical and logical files for an example of data sorted using the DESCEND keyword.

This keyword has no parameters.

Example

The following example shows how to specify the DESCEND keyword for a logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          K ITEM
00020A          K BALDUE                  DESCEND
     A
```

Related reference

SIGNED (Signed) keyword for physical and logical files
If this key field-level keyword is in effect, when sequencing the values associated with this numeric key field, the operating system considers the signs of the values (negative versus positive values).

DFT (Default) keyword—physical files only

You can use this field-level keyword to specify a default value for a field.

The format of the keyword is:

```
DFT('value' | numeric-value | X'hexadecimal-value' | *NULL)
```

Without this keyword, character and hexadecimal fields are set to blanks as default and numeric fields are set to zeros as default. However, if you specify the ALWNULL keyword for the field, then the character, hexadecimal, and numeric fields are set to the null value as default.

The following rules apply to the specified value:

- If the field being defined is a character field, specify a character constant, hexadecimal value, or *NULL. Specify character strings within single quotation marks. If the field is variable length (VARLEN), then the length of the string must be less than or equal to the allocated length.

Specify hexadecimal values as an X followed by a combination of the digits 0 through 9 and the letters A through F. Enclose the combination in single quotation marks. The number of hexadecimal digits in single quotation marks must be exactly twice the specified length of the field. If the field is variable length (VARLEN), then the number of hexadecimal digits in single quotation marks must be exactly twice the allocated length.

- If the field being defined is a hexadecimal field, specify a character constant, hexadecimal value, or *NULL.

  Note: If a character constant is specified, the hexadecimal representation of the character constant is the default value.

  Specify character strings within single quotation marks. If the field is variable length (VARLEN), then the length of the string must be less than or equal to the allocated length.

  Specify hexadecimal values as an X followed by a combination of the digits 0 through 9 and the letters A through F. Enclose the combination in single quotation marks. The number of hexadecimal digits in single quotation marks must be exactly twice the specified length of the field. If the field is variable length (VARLEN), then the number of hexadecimal digits in single quotation marks must be exactly twice the allocated length.
- If you are defining a numeric field, specify a numeric value (digits 0 through 9 specified without single quotation marks) or *NULL. For a value other than zero in positions 36 and 37, specify the decimal character with a numeric constant in the appropriate position in the DDS.
- If you specify *NULL, then you must also specify the ALWNULL keyword on the field.
- If you do not specify any value (DFT('')), this indicates a default of a 0 length string and is valid only when the field is variable length (the VARLEN keyword must also be specified).
- If you are defining a date field, specify a valid date in the same format specified on the DATFMT keyword and use the same separator as specified on the DATSEP keyword. For example, `DFT('12/15/05')` is the default value if *MDY is specified for DATFMT and '/' is specified for DATSEP.

  If the DFT keyword is not specified, the default value is the current date.
- If you are defining a time field, specify a valid time in the same format specified on the TIMFMT keyword and use the same separator as specified on the TIMSEP keyword. For example, `DFT('11.00.00')` is the default value if *ISO is specified for TIMFMT. The default separator for *ISO is a period (.).

  If the DFT keyword is not specified, the default value is the current time.
- If you are defining a timestamp field, you must specify the default value in the following format:

  ```
  DFT('YYYY-MM-DD-HH.MM.SS.UUUUUU')
  ```

  If the DFT keyword is not specified, the default value is the current time.

The value specified is assigned to the field in the following cases:

- When the program does an output operation to a logical file based on this physical file and the record format in the logical file does not name this field.
- When you use the Initialize Physical File Member (INZPFM) command for a member in this file.
- When you use the Copy File (CPYF) command with FMTOPT(*MAP) specified and a field in the to-file is not in the from-file.

The specified value is supplied to the program when the program does an input operation to a join logical file and all of the following situations are true:

- You specify the JDFTVAL keyword for the join logical file.
- The file being defined is specified as a secondary file in the join logical file.
- When the input operation occurs and the link to the secondary file produces no records.

This keyword does not affect the physical file on input operations.

Example

The following example shows how to specify the DFT keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1
00020A            CHARFLD1      20A         DFT('Sample field')
00030A            CHARFLD2       5A         DFT(X'D985955185')
00040A            HEXFLD1        3H         DFT('ABC')
00050A            HEXFLD2        3H         DFT(X'C1C2C3')
00060A            NUMFLD1        5S 0       DFT(99999)
00070A            NUMFLD2        5S 2       DFT(999.99)
00080A            NUMFLD3        5S 2       DFT(999)
00090A            NUMFLD4        5S 2       DFT(*NULL)
00100A                                      ALWNULL
00110A            NUMFLD5        5S 2       DFT(999.99)
00120A                                      ALWNULL
00130A            DATFLD1         L         DATFMT(*MDY) DATSEP('-')
00140A                                      DFT('12-31-05')
00150A            TIMFLD1         T         DFT('11.15.00')
     A
```

The default value for CHARFLD1 is 'Sample field'. The default value for CHARFLD2 is hexadecimal D985955185. The default value for HEXFLD1 is C1C2C3 (the hexadecimal representation of the character constant). The default value for HEXFLD2 is C1C2C3. The default value for NUMFLD1 is 99999 (no decimal character is required because the field has zero decimal positions). The default value for NUMFLD2 is 999.99. The default value for NUMFLD3 is 999 (no decimal character is required if you do not need to specify decimal values). The default value for NUMFLD4 is the null value (ALWNULL is a required keyword for the field if DFT(*NULL) is specified). The default value for NUMFLD5 is 999.99; the field also allows the null value. The default value for DATFLD1 is 12-31-05. The default value for TIMFLD1 is 11.15.00 (*ISO format).

Related reference

Usage for physical and logical files (position 38)
You use this field to specify that a named field is to be an input-only, both (both input and output are allowed), or neither (neither input nor output is allowed) field.

JDFTVAL (Join Default Values) keyword—join logical files only
The JDFTVAL keyword is valid only for join logical files. If this keyword is in effect, the system provides default values for fields when a join to a secondary file does not produce any records.

DIGIT (Digit) keyword for physical and logical files

If this key field-level keyword is in effect, only the digit portion (farthest right 4 bits) of each byte of the key field is used when the system constructs a value associated with this key field. The zone portion is zero-filled.

This keyword has no parameters.

The DIGIT keyword is applied against the entire key field (not just a position within the field). It is valid only for character, hexadecimal, or zoned decimal type fields.

You cannot use this keyword with the ABSVAL, SIGNED, or ZONE keywords.

If you specify DIGIT for a key field, the value of the field is treated as a string of unsigned binary data, rather than signed data, which is the default for zoned decimal fields.

Example

The following example shows how to specify the DIGIT keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00040A          K ORDTYP                    DIGIT
     A
```

If ORDTYP is a 3-byte field, the values of the field for three different records might be as follows:

Values	Hexadecimal	Digits used for key
C4J	C3F4D1	341
CMA	D3D4C1	341
3D1	F3C4F1	341

DYNSLT (Dynamic Select) keyword—logical files only

You can use this file-level keyword to indicate that the selection and omission tests specified in the file (using select/omit specifications) are done at processing time.

This keyword specifies dynamic select/omit rather than access path select/omit. This keyword has no parameters.

When your program does input operations to a logical file with the DYNSLT keyword specified, all the records in the associated physical file are tested by the system to see if they satisfy the select/omit values. Only those records that satisfy the values are supplied to your program. The testing of each record can result in slower I/O performance, but can be more efficient than maintaining an access path for the file. This is particularly likely for files read only occasionally, especially when the physical files they are based on are updated frequently. Using dynamic select/omit is probably also more efficient for files with a high percentage of selected records.

In keyed sequence access files, an access path is created at file creation time and is maintained for the file according to the MAINT parameter on the Create Logical File (CRTLF) or Change Logical File (CHGLF) command. The DYNSLT keyword does not affect the maintenance of access paths for keyed sequence access files.

For all single-format logical files with a DYNSLT keyword, you do not need to specify key fields in order to specify select/omit fields. However, for all multiple-format logical files with a DYNSLT keyword, you do need to specify at least one key field. You can specify *NONE for this key field.

You must use the DYNSLT keyword when you want to select or omit fields and any of the following situations are true:

- The logical file has arrival sequence (no key fields are specified). See example 1 in this topic.
- The logical file is a join logical file with the JDFTVAL keyword specified.
- The logical file is a join logical file, select/omit fields come from more than one of the physical files the logical file is based on, and one of the following conditions is true:
  - The select/omit fields are on the same select or omit statement. See example 3 in this topic.
  - The select/omit fields are on a mixture of select and omit statements. See example 4 in this topic.
  - The select/omit fields are on select statements that are grouped together by OR.
  - The select/omit fields are on omit statements that are grouped together by AND.

You cannot specify the DYNSLT keyword with the REFACCPTH keyword.

For a join logical file, the select/omit fields can occur in any of the physical files specified on the JFILE keyword. Use the JREF keyword in join logical files to qualify the origin of the field and resolve any ambiguities.

Examples

The following examples show how to specify the DYNSLT keyword.

Example 1

The following example shows how to specify dynamic select with arrival sequence.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                DYNSLT
00020A          R RECORD1             PFILE(PF1)
```

```
00030A            FLD1
00040A            FLD2
00050A          S FLD1                    COMP(GT 2)
```

The DYNSLT keyword is required because there are no key fields.

The logical file supplies records to your program in arrival sequence. Assume that physical file PF1 has the following records:

FLD1
FLD2

1
aaaa

2
dddd

3
jjjj

4
bbbb

As your program does input operations, the system tests the first two records according to the select/omit values, but does not supply them to your program. Your program only sees the last two records:

FLD1
FLD2

3
jjjj

4
bbbb

Example 2

The following example shows how to specify dynamic select with keyed sequence access path.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                  DYNSLT
00020A          R RECORD1               PFILE(PF1)
00030A            FLD1
00040A            FLD2
00050A          K FLD1
00060A          S FLD2                  COMP(GT 'bbbb')
     A
```

In this example, the DYNSLT keyword is not required. The logical file supplies records to your program in keyed sequence. Assume that physical file PF1 has the following records:

FLD1
FLD2

1
aaaa

2
dddd

3
jjjj

4
bbbb

When your program requests a record, the system tests the value of FLD2 for that record according to the select/omit values. Your program only sees the following records:

FLD1
FLD2

2
dddd

3
jjjj

Example 3

The following example shows how to specify a join logical file with select/omit comparing fields from two physical files.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                DYNSLT
00020A          R RECORD1             JFILE(PF1 PF2)
00030A          J                     JFLD(FLD1 FLD3)
00040A            FLD1                JREF(PF1)
00050A            FLD2                JREF(PF1)
00060A            FLD3                JREF(PF2)
00070A            FLD4                JREF(PF2)
00080A          S FLD1                COMP(GT FLD4)
     A
```

FLD1 and FLD2 come from the primary file (PF1), and FLD3 and FLD4 come from the secondary file (PF2). The select specification compares FLD1 from the primary file with FLD4 from the secondary file. Therefore, the DYNSLT keyword is required.

Example 4

The following example shows how to specify a join logical file with select and omit using fields from more than one physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                DYNSLT
00020A          R JREC                JFILE(PF1 PF2)
00030A          J                     JOIN(PF1 PF2)
00040A                                JFLD(FLD1 FLD2)
00050A            FLD1                JREF(PF1)
00060A            FLD2                JREF(PF1)
00070A            FLD3                JREF(PF2)
00080A          K FLD1
00090A          S FLD1                COMP(GT 0)
00100A          O FLD3                COMP(GT 4)
     A
```

FLD1 and FLD3 come from different physical files and are specified in a mixture of select and omit statements. Therefore, the DYNSLT keyword is required.

EDTCDE (Edit Code) and EDTWRD (Edit Word) keywords for physical and logical files

You can use these field-level keywords to specify editing for the field you are defining when the field is referenced later during display or printer file creation. The EDTCDE and EDTWRD keywords do not affect the physical or logical file.

The format of the EDTCDE keyword is:

```
EDTCDE(edit-code [* | floating-currency-symbol])
```

The format of the EDTWRD keyword is:

```
EDTWRD('edit-word')
```

When defining an input-capable field in a display file, refer to the field you are defining by specifying the letter R in position 29 and the REF or REFFLD keyword. At display file creation, the operating system

copies the EDTCDE or EDTWRD keyword and other field attributes from the field in the physical or logical file into the field in the display file. You can override the EDTCDE or EDTWRD keyword by specifying new editing keywords in the display or printer file. Specifying the DLTEDT keyword in the display or printer file deletes all editing for the field.

You cannot specify the EDTCDE or EDTWRD keyword on a floating-point field (F in position 35) or a hexadecimal field (H in position 35). Do not specify the EDTCDE or EDTWRD keyword on a date, time, or timestamp field (L, T, or Z in position 35).

The rules for specifying these keywords in a physical or logical file are the same as the rules for a display file.

Example

The following example shows how to specify the EDTCDE and EDTWRD keywords for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RECORD
     A
     A            PRICE          5  2       EDTCDE(J)
     A
     A            SALES          7  2       EDTCDE(K $)
     A
     A            SALARY         8  2       EDTCDE(1 *)
     A
     A            BALANCE        7  2       EDTWRD('$    0.  &CR')
     A
     A            DATE           6  0       EDTCDE(Y)
     A
```

The fields PRICE, SALES, SALARY, and DATE have editing specified. No new editing needs to be specified when they are referred to by a display or printer file. This standardizes the editing of these fields for applications that refer to these fields.

Related information

Reference for display files (position 29)
EDTCDE (Edit Code) keywords for display files
EDTWRD (Edit Word) keywords for display files

FCFO (First-Changed First-Out) keyword for physical and logical files

You can use this file-level keyword to specify that if records with duplicate key values are retrieved from the same physical or logical file member, the record with the key value that was changed first is the first record retrieved. This is in a first-changed first-out (FCFO) order.

This keyword has no parameters.

FCFO is not allowed with an FIFO, LIFO, UNIQUE, or REFACCPTH keyword.

If you do not specify FCFO, LIFO, FIFO, or UNIQUE, records with duplicate key values are retrieved in first-in first-out (FIFO), last-in first-out (LIFO), or first-changed first-out (FCFO) order, but the order in which they are retrieved is not guaranteed.

With the FCFO keyword, the records are ordered by the time the record key value is changed. With the FIFO and LIFO keywords, the records are ordered by the relative record number.

At least one key field must be specified in the file containing the FCFO keyword. The FCFO keyword is not valid when you specify FILETYPE(*SRC) on the Create Physical File (CRTPF) or Create Logical File (CRTLF) command.

Example

The following example shows how to specify the FCFO keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                 FCFO
```

```
00020A          R CUSREC                    TEXT('CUSTOMER RECORD')
00030A            CUSNAMEF      10A
00040A            CUSNAMEM       1A
00050A            CUSNAMEL      10A
00060A          K CUSNAMEL
     A
```

FIFO (First-In First-Out) keyword for physical and logical files

You can use this file-level keyword to specify that if records with duplicate key values are retrieved from the same physical or logical file member, they are to be retrieved in a first-in first-out (FIFO) order.

This keyword has no parameters.

FIFO is not allowed with an first-changed first-out (FCFO), last-in first-out (LIFO), UNIQUE, or REFACCPTH keyword.

If you do not specify FCFO, LIFO, FIFO, or UNIQUE, records with duplicate key values are retrieved in FIFO, LIFO, or FCFO order, but the order in which they are retrieved is not guaranteed.

At least one key field must be specified in a file containing the FIFO keyword. The FIFO keyword is not valid when you specify FILETYPE(*SRC) on the Create Physical File (CRTPF) or Create Logical File (CRTLF) command.

Example

The following example shows how to specify the FIFO keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                  FIFO
00020A          R CUSREC                TEXT('CUSTOMER RECORD')
00030A            CUSNAMEF      10A
00040A            CUSNAMEM       1A
00050A            CUSNAMEL      10A
00060A          K CUSNAMEL
     A
```

FLTPCN (Floating-Point Precision) keyword for physical and logical files

You can use this field-level keyword to specify the precision of a floating-point field.

The format of the keyword is:

```
FLTPCN(*SINGLE | *DOUBLE)
```

where *SINGLE is single precision and *DOUBLE is double precision. This keyword is valid for floating-point fields only (data type F).

If you do not specify the FLTPCN keyword, the default is single precision. A single precision field can be up to 9 digits; a double precision field can be up to 17 digits. If you specify a field length greater than 9 (single precision) or 17 (double precision), an error message is sent and the file is not created.

Example

The following example shows how to specify the FLTPCN keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00090A            FIELDA        17F 4       FLTPCN(*DOUBLE)
     A
```

FIELDA is a floating-point field with double precision.

FORMAT (Format) keyword for physical and logical files

You can use this record-level keyword to specify that this record format is to share the field specifications for a previously defined record format. The name of the record format you are defining must be the name of the previously defined record format.

The format of the keyword is:

```
FORMAT([library-name/]database-file-name)
```

The database-file-name parameter is required. It is the name of the physical or logical file from which the previously defined record format is taken.

The library-name is optional. If you do not specify the library-name, the library list (*LIBL) in effect at file creation time is used.

If you specify the FORMAT keyword, you cannot specify field specifications for this record format. Specify key specifications and, if necessary, select/omit specifications if you want them to be in effect for this file. (They can be the same as or different from the previously defined record format.)

The FORMAT keyword is not valid in join logical files and you cannot specify a join logical file as the parameter value on the FORMAT keyword.

If the database file from which you are using the record format is deleted, the record format remains in existence as long as some file is using the record format. For example, RECORD in FILE2 uses the FORMAT keyword to share the specifications of RECORD in FILE1. Both files have been created. If FILE1 is deleted and then re-created with different DDS, RECORD still exists in FILE2. It can be referred to for the original record format by other files using the FORMAT keyword.

You cannot specify a distributed data management (DDM) file on this keyword.

Example

The following example shows how to specify the FORMAT keyword for a logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD                    PFILE(FILE2)
00020A                                      FORMAT(FILE1)
     A
```

The record format for this logical file is the same as the previously specified record format in file FILE1. The name of this record format (RECORD) must be the same as the name of the record format in FILE1.

Related concepts

Record format
When you specify R in position 17, the name specified in positions 19 through 28 is a record format name.

JDFTVAL (Join Default Values) keyword—join logical files only

The JDFTVAL keyword is valid only for join logical files. If this keyword is in effect, the system provides default values for fields when a join to a secondary file does not produce any records.

This keyword has no parameters.

The default values for the system are blanks for character and hexadecimal fields and zeros for numeric fields. You can change the default for specific fields by specifying the DFT keyword for the fields in the physical file.

If you specify JDFTVAL, your program retrieves records for which a secondary file does not have a corresponding record. If you do not specify JDFTVAL, a record in the primary file for which there is no corresponding record in a secondary file is skipped.

If you are joining three or more files, and you specify the JDFTVAL keyword for fields used as join fields, default values of fields missing in secondary files are used in the same way that a field value is used. For example, records are selected and omitted based on the default value. Also, if this field is used as a join

field to join to other secondary files, records from the other secondary files are returned to your program based on the default value.

Example

The following example shows how to specify the JDFTVAL keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                  JDFTVAL
00020A          R RECORD1               JFILE(PF1 PF2)
00030A          J                       JOIN(PF1 PF2)
00040A                                  JFLD(NAME NAME)
00050A            NAME                  JREF(1)
00060A            ADDR
00070A            BAL
     A
```

PF1 is the primary file and PF2 is a secondary file. Assume that PF1 and PF2 have the following records:

PF1 NAME	ADDR	PF2 NAME	BAL
Anne	120 1st St.	Anne	5.00
Doug	40 Pillsbury	Doug	6.50
Mark	2 Lakeside Dr.	Sue	2.00
Sue	120 Broadway		

With JDFTVAL specified in the join logical file, the program reads the following records (shown in arrival sequence):

NAME	ADDR	BAL
Anne	120 1st St.	5.00
Doug	40 Pillsbury	6.50
Mark	2 Lakeside Dr.	0.00
Sue	120 Broadway	2.00

Without JDFTVAL specified in the join logical file, the program can read only three records (no record is found for Mark).

In this example, if you specified JREF(2) instead of JREF(1), the records returned to the program might be different, as follows:

NAME	ADDR	BAL
Anne	120 1st St.	5.00
Doug	40 Pillsbury	6.50
	2 Lakeside Dr.	0.00
Sue	120 Broadway	2.00

Related reference

DFT (Default) keyword—physical files only
You can use this field-level keyword to specify a default value for a field.

JDUPSEQ (Join Duplicate Sequence) keyword—join logical files only

You can use this join-level keyword to specify the order in which records with duplicate join fields are presented when your program reads a join logical file.

The format of the keyword is:

```
JDUPSEQ(sequencing-field-name [*DESCEND])
```

This keyword has no effect on the ordering of unique records. If you do not specify the keyword, the system does not guarantee the order in which records with duplicate join fields are presented.

If more than one JDUPSEQ keyword is specified in one join specification, the order in which you specify the JDUPSEQ keywords determines the order of presentation of duplicate records. This is similar to specifying an additional key field, in that it determines the order in which records with duplicate keys are presented.

This keyword is valid only for join logical files.

In a single join specification, the total length of fields specified as *to* fields on the JFLD keyword and fields specified on the JDUPSEQ keyword cannot exceed 120 bytes.

The sequencing field name must be a field that (1) exists in the *to* file for this join specification and (2) has not been specified as a *to* field on the JFLD keyword for this join specification. The sequencing field name can be a concatenated field or a SST field. The sequencing field name need not be specified in the record format for the join logical file.

Optionally, you can specify *DESCEND to change the order in which duplicate records are presented. Without *DESCEND, duplicate records are presented in the following default sequences:

- Ascending signed order for a numeric sequencing field
- Ascending order for a character sequencing field

Examples

The following example shows how to specify the JDUPSEQ keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JREC                      JFILE(PF1 PF2)
00020A          J                           JOIN(PF1 PF2)
00030A                                      JFLD(NAME1 NAME2)
00040A                                      JDUPSEQ(PHONE)
00050A            NAME1
00060A            ADDR
00070A            PHONE
```

This example assumes that PF1 and PF2 have the following records:

PF1 NAME1	ADDR	PF2 NAME2	TELEPHONE
Anne	120 1st St.	Anne	555-1111
Doug	40 Pillsbury	Anne	555-6666
Mark	2 Lakeside Dr.	Anne	555-2222
		Doug	555-5555

There are three records for Anne in PF2, showing three telephone numbers. With JDUPSEQ specified as shown, the records are returned as follows:

NAME	ADDR	TELEPHONE
Anne	120 1st St.	555-1111
Anne	120 1st St.	555-2222
Anne	120 1st St.	555-6666
Doug	40 Pillsbury	555-5555

The JDUPSEQ keyword only affects the order of records when duplicates exist.

The following example assumes that the logical file is based on the same physical files as example 1. There are three records for Anne in PF2, showing three telephone numbers.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JREC                      JFILE(PF1 PF2)
00020A          J                           JOIN(PF1 PF2)
00030A                                      JFLD(NAME1 NAME2)
00040A                                      JDUPSEQ(PHONE *DESCEND)
00050A            NAME1
00060A            ADDR
00070A            PHONE
     A
```

When you specify JDUPSEQ with *DESCEND, the records are returned as follows:

NAME1	ADDR	TELEPHONE
Anne	120 1st St.	555-6666
Anne	120 1st St.	555-2222
Anne	120 1st St.	555-1111
Doug	40 Pillsbury	555-5555

The list shows Anne's telephone numbers in descending order.

JFILE (Joined Files) keyword—join logical files only

You can use this record-level keyword to identify the physical files that contain the data to be accessed through the join logical file you are defining.

The format of the keyword is:

```
JFILE([library-name/]physical-file-name [..256])
```

This keyword is similar to the PFILE keyword except that it identifies this file as a join logical file. The JFILE keyword is not allowed with the PFILE keyword.

The JFILE keyword is required at the record level in a join logical file. The JFILE keyword requires a minimum of two physical file names. You can specify the same file name more than once.

The first file is called the primary file, which is the file from which the join begins. All other files are called secondary files. Up to 255 secondary files can be specified (256 total files on the JFILE keyword).

Distributed data management (DDM) files are allowed on the JFILE keyword only when the logical file is being created on a remote system.

The following considerations apply to the order in which you specify physical files on the JFILE keyword:

- If the physical files have a different number of records, specify physical files with fewer records toward the left on the JFILE keyword. The primary file should have as many or fewer records than the secondary files. This can improve performance when reading files.
- Primary and secondary files specified in join specifications must be in a specific order. This order depends on the order in which the files are specified on the JFILE keyword. See Example 3 in JOIN (Join) keyword—join logical files only.
- The JOIN and JREF keywords can use relative file numbers to identify files specified by the JFILE keyword. The first file specified on the JFILE keyword has relative file number 1, the second file has relative file number 2, and so on up to 256. If you use relative file numbers instead of file names on the JOIN and JREF keywords, the order of files on the JFILE keyword can affect the way the JOIN and JREF keywords are specified.

Note: If the names in the physical file are not unique, you must specify relative file numbers.

Examples

The following examples show how to specify the JFILE keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JREC                      JFILE(PF1 PF2)
00020A          J                           JOIN(PF1 PF2)
00030A                                      JFLD(NAME1 NAME2)
     A
```

In the join logical file, PF1 is the primary file and PF2 is the secondary file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JREC                      JFILE(MYLIBA/PHYSICAL1 +
00020A                                      MYLIBB/PHYSICAL2 MYLIBC/PHYSICAL3)
00030A          J                           JOIN(1 2)
00040A                                      JFLD(FIELD1 FIELD2)
00050A          J                           JOIN(1 3)
00060A                                      JFLD(FIELD1 FIELD2)
     A
```

In the join logical file, file PHYSICAL1 in library MYLIBA is the primary file. File PHYSICAL2 in library MYLIBB and file PHYSICAL3 in library MYLIBC are secondary files.

Related reference

JOIN (Join) keyword—join logical files only
You can use this join-level keyword to identify which pair of files are joined by the join specification in which you specify this keyword.

Related information

Distributed database programming

JFLD (Joined Fields) keyword—join logical files only

You can use this join-level keyword to identify the *from* and *to* fields whose values are used to join physical files in a join logical file. These fields are both referred to as join fields.

The format of the keyword is:

```
JFLD(from-field-name to-field-name)
```

The join fields must correspond to fields in the physical files identified on the JOIN keyword for this join specification. The name you specify on the JFLD keyword must be the same as the name specified in the physical file unless it was renamed in the join logical file. If you do not specify a JOIN keyword, then the JFILE keyword is used.

This keyword is valid only for join logical files.

At least one JFLD keyword is required for each join specification. A join specification is identified by J in position 17. Because at least one join specification is required in a join logical file, you must have at least one JFLD keyword specified in a join logical file.

These fields need not also be specified as fields in the record format for a join logical file.

To specify additional join fields to use when joining physical files, specify more than one JFLD keyword.

The field names you specify on the JFLD keyword must be specified either at the field level in the join record format or in one of the physical files, which are specified on the JFILE keyword.

The IBM i operating system uses the following search order to match join field names with defined fields:

1. Fields specified in the join logical file at the field level in positions 19 through 28.

   Note: Fields that specify the CONCAT, RENAME, or SST keyword are valid as join fields; fields that are specified on CONCAT, RENAME, or SST keyword cannot be join fields.
2. Fields in the physical file specified on the JOIN keyword.

The rules for specifying join fields are as follows:

- The *from* field must be found in the *from* file specified on the JOIN keyword.
- The *to* field must be found in the *to* file specified on the JOIN keyword.
- Join fields are not required to be defined in the join record format.
- *From* and *to* fields must have the same field attributes (length, data type, and decimal positions) but need not have the same name. When the joined fields in the physical files have different definitions, you must redefine one or both fields. If you redefine fields, there is a possibility of data conversion errors.

  Note: Character fields need not have the same length. The shorter join field is padded with blanks to equal the length of the longer join field.
- In a single join specification, the total length of fields specified as *to* fields on the JFLD keyword and fields specified on the JDUPSEQ keyword can be up to 120 bytes.
- Binary character fields can be joined only with other binary character fields.

Examples

The following examples show how to specify the JFLD keyword.

Example 1

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JREC                      JFILE(PF1 PF2)
00020A          J                           JOIN(PF1 PF2)
00030A                                      JFLD(NAME1 NAME2)
     A
```

In the join logical file, the JFLD keywords specify that NAME1 in physical file PF1 is used to join to NAME2 in physical file PF2.

Example 2

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JREC                      JFILE(PF1 PF2)
00020A          J                           JOIN(PF1 PF2)
00030A                                      JFLD(NAME1 NAME2)
00040A                                      JFLD(ADDR1 ADDR2)
     A
```

In the join logical file, the JFLD keywords specify that NAME1 and ADDR1 in physical file PF1 are used to join to NAME2 and ADDR2 in physical file PF2.

Related reference

Length for physical and logical files (positions 30 through 34)
You use these positions to specify the length of a physical or logical file field.

Data type for physical and logical files (position 35)
For a physical file, you use this position to specify the data type of the field within the database. You specify data type in a logical file only to override or change the data type of the corresponding field in the physical file on which this logical file is based.

Decimal positions for physical and logical files (positions 36 and 37)
You use these positions to specify the decimal placement within a packed decimal, zoned decimal, binary, or floating-point field.

Usage for physical and logical files (position 38)

You use this field to specify that a named field is to be an input-only, both (both input and output are allowed), or neither (neither input nor output is allowed) field.

JOIN (Join) keyword—join logical files only

You can use this join-level keyword to identify which pair of files are joined by the join specification in which you specify this keyword.

The format of the keyword is:

```
JOIN(from-file to-file)
```

This keyword is valid only for join logical files.

You can use file names or relative file numbers to indicate which files are to be joined. You must specify a relative file number if the same file is specified more than once on the JFILE keyword.

If you specify file names, you must select files that you have specified only once on the JFILE keyword. On each JFILE keyword, the *from* file must occur before the *to* file.

If you specify numbers, they correspond to the files specified on the JFILE keyword. The following values are valid:

File
Valid values

From-file number
1 through 255

To-file number
2 through 256

The *from-file* number must always be less than the *to-file* number.

Special rules apply to the order in which you specify *from* and *to* files. See example 3 in this topic for details.

In a join logical file, each secondary file can be a *to* file only once.

Join specifications for physical and logical files

To describe a join specification, follow these steps:

1. Specify J in position 17 immediately after the record level (before the first field name in positions 19 through 28). J in position 17 indicates the beginning of a join specification.
2. Specify the JOIN keyword. The JOIN keyword is optional when only two files are specified on the JFILE keyword. When more than two physical files are specified on the JFILE keyword, one JOIN keyword is required for each secondary file.
3. Specify the JFLD keyword at least once for each join specification.
4. The end of the join specification is indicated by another J in position 17 or by a field name specified in positions 19 through 28.

There must be one join specification for each secondary file specified on the JFILE keyword. Therefore, at least one join specification is required in a join logical file.

You can specify the JOIN keyword only once within a join specification.

Examples

The following examples show how to specify the JOIN keyword.

Example 1

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                  JFILE(PFA PFB PFC)
```

```
00020A          J                           JOIN(PFA PFB)
00030A                                      JFLD(NAME1 NAME2)
00040A          J                           JOIN(PFA PFC)
00050A                                      JFLD(NAME1 NAME3)
00060A            NAME1
     A
```

In this example, PFA is joined to PFB and also to PFC.

Example 2

The following example shows how to specify JOIN using relative file numbers.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   JFILE(PFA PFB PFC)
00020A          J                           JOIN(1 2)
00030A                                      JFLD(NAME1 NAME2)
00040A          J                           JOIN(1 3)
00050A                                      JFLD(NAME1 NAME3)
00060A            NAME1
     A
```

Example 2 is equivalent to example 1. PFA is the first physical file specified on the JFILE keyword and has relative file number 1. PFB and PFC are the second and third files specified on the JFILE keyword. PFB has relative file number 2, and PFC relative file number 3.

Example 3

The following example shows the order of associated physical files.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R J3                        JFILE(VENDORS PARTS PARTWARE +
00020A                                            WAREHOUSE
1
00030A          J                           JOIN(1   2)   2
00040A                                      JFLD(VNBR VNUM)
00050A          J                           JOIN(2   3)   3
00060A                                      JFLD(PNBR PNBR)
00070A          J                           JOIN(3   4)   3
00080A                                      JFLD(WNBR WNBR)
00090A            VNAME
00100A            VAD1
00110A            VAD2
00120A            PNBR                      JREF(2)
00130A            WNBR                      JREF(4)
00140A            BIN
00150A            QOH
     A
```

The join logical file in this example is based on four physical files. The VENDORS file, which is specified first on the JFILE keyword, is the primary file and has relative file number 1. The PARTS, PARTWARE, and WAREHOUSE files are secondary files. PARTS has relative file number 2, PARTWARE relative file number 3, and WAREHOUSE relative file number 4.

Notice the pattern of numbers specified on the JOIN keywords:

1

The first parameter value on the first JOIN keyword (the first *from* file) must be the primary file.

2

The second parameter values specified on the JOIN keywords (*to* files) must reflect the same order as the secondary files on the JFILE keyword. If file names are specified instead of relative file numbers, they should be specified in the following order:

```
J    JOIN(VENDORS PARTS)
J    JOIN(PARTS PARTWARE)
J    JOIN(PARTWARE WAREHOUSE)
```

3

On each JOIN keyword, the *from* and *to* files must be specified in ascending order.

Note: A file can be specified as a *from* file more than once. For example, the parameters on the JOIN keywords before might be specified as follows:

```
J    JOIN(1 2)
J    JOIN(2 3)
J    JOIN(2 4)
```

However, a file can be specified as a *to* file only once.

Related reference

Type of name or specification for physical and logical files (position 17)
For physical files, type a value in this position to identify the type of name. For logical files, type a value to identify the type of specification. If you specify a name type, the name is specified in positions 19 through 28.

JFILE (Joined Files) keyword—join logical files only
You can use this record-level keyword to identify the physical files that contain the data to be accessed through the join logical file you are defining.

JREF (Join Reference) keyword—join logical files only

You can use this field-level keyword in join logical files for fields whose names are specified in more than one physical file. This keyword identifies which physical file contains the field.

The format of the keyword is:

```
JREF(file-name | relative-file-number)
```

You can specify either the physical file name or its relative file number. If a physical file is named twice on the JFILE keyword, then you must specify the relative file number. The relative file number corresponds to the physical file name specified on the JFILE keyword. For example, specifying JREF(1) associates a field with the first physical file specified on the JFILE keyword. Specifying JREF(2) associates a field with the second physical file specified on the JFILE keyword. See example 2 in this topic.

This keyword is valid only in a join logical file.

Join logical files are based on two or more physical files (up to 256). Field names specified in the record format in a join logical file must uniquely identify only one field from the physical files on which the join logical file is based. For example, if the join logical file is based on two physical files, and each physical file has the field named NAME, you must specify the JREF keyword to identify which physical file the field comes from.

When a field name is unique among the physical files specified on the JFILE keyword, this keyword is optional. For example, if the join logical file is associated with two physical files, and only one of the physical files has a field named NAME1, you do not need to specify the JREF keyword.

If the join logical file is associated with only one physical file (the JFILE keyword names the same file twice), you must specify the JREF keyword on every field.

Example 1

The following examples show how to specify the JREF keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JOINREC                   JFILE(PFA PFB PFC)
00020A              :
00030A              :
00040A              :
00050A            NAME                      JREF(PFB)
     A
```

In this example, the JREF keyword is specified with the file name, and NAME occurs in both PFA and PFB. Specifying JREF (PFB) associates this field with PFB.

Example 2

The following example shows how to use the file reference numbers to specify JREF.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R JOINREC                   JFILE(PFA PFB PFC)
00020A               :
00030A               :
00040A               :
00050A            NAME                      JREF(2)
     A
```

Example 2 is equivalent to example 1. In example 2, NAME occurs in both PFA and PFB. Specifying JREF(2) associates this field with PFB (the second of the physical files specified on the JFILE keyword).

LIFO (Last-In First-Out) keyword for physical and logical files

You can use this file-level keyword to specify that records with duplicate key values from the same physical file member are retrieved in a last-in first-out (LIFO) order.

This keyword has no parameters.

LIFO is not allowed with an FCFO, FIFO, UNIQUE, or REFACCPTH keyword.

If you do not specify first-changed first-out (FCFO), first-in first-out (FIFO), LIFO, or UNIQUE, records with duplicate key values are retrieved in FIFO, LIFO, or FCFO order, but the order in which they are retrieved is not guaranteed.

At least one key field must be specified in a file containing the LIFO keyword. The LIFO keyword is not valid when you specify FILETYPE(*SRC) on the Create Physical File (CRTPF) or Create Logical File (CRTLF) command.

Example

The following example shows how to specify the LIFO keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                      LIFO
00020A          R CUSREC                    TEXT('CUSTOMER RECORD')
00030A            CUSNAMEF      10A
00040A            CUSNAMEM       1A
00050A            CUSNAMEL      10A
00060A          K CUSNAMEL
     A
```

NOALTSEQ (No Alternative Collating Sequence) keyword for physical and logical files

You can use this key field-level keyword to specify that the ALTSEQ keyword specified at the file level does not apply to this key field.

If you specify ABSVAL or SIGNED for a key field, NOALTSEQ is automatically in effect, whether the NOALTSEQ keyword is specified for that key field.

This keyword has no parameters.

Example

The following example shows how to specify the NOALTSEQ keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                      ALTSEQ(TABLELIB/TABLE1)
00020A          R DSTR
00030A             :
00040A             :
00050A            CODE           1
00060A            NAME          20
00070A             :
```

```
00080A             :
00090A          K CODE
00100A          K NAME                      NOALTSEQ
     A
```

Records with the record format DSTR are sequenced by the composite keys CODE and NAME. CODE is sequenced by the alternative collating sequence (TABLE1 in TABLELIB). NAME is sequenced by the EBCDIC collating sequence. NOALTSEQ prevents the sequence of the NAME field from being altered.

PFILE (Physical File) keyword—logical files only

You can use this record-level keyword to identify the physical files that contain the data to be accessed through the record format that you are defining.

The format of the keyword is:

```
PFILE([library-name/]physical-file-name [.32])
```

The PFILE keyword is required on every record format in a simple or multiple format logical file. This keyword is similar to the JFILE keyword except that it identifies this file as a simple or multiple format logical file; the PFILE keyword is not allowed with the JFILE keyword. Up to 32 physical file names can be specified on PFILE keywords in a logical file. If the maximum is being used, 32 physical file names can be specified on one record format (using one PFILE keyword) or 32 physical file names can be distributed among 32 record formats; or, file names can be unevenly distributed among record formats. In any case, the maximum number of physical file names allowed is 32. For restrictions on specifying multiple physical files when creating a logical file, see the appropriate high-level language manual.

For each physical-file-name, a library-name is optional. If the library-name is omitted, the library list (*LIBL) that is in effect at file creation time is used.

If you specify more than one physical file name for one record format in a multiple format logical file, all fields in the record format for the logical file must exist in all physical files specified. This type of file cannot be externally described in RPG because it results in duplicate format names. If your program requires access to fields that occur in one or more of the physical files specified on the PFILE keyword, but not in all of them, you can do one of the following tasks:

- Specify a join logical file. If you do this, use the JFILE keyword instead of the PFILE keyword.
- Specify a separate logical file record format that includes fields not in the first physical file.

  For instance, if FLD1 and FLD2 occur in physical files PF1, PF2, and PF3, but FLD3 occurs only in PF3, you cannot specify FLD3 in a logical file record format based on PF1 and PF2. To provide access to FLD3, either specify a second logical file record format that includes FLD3 or use a join logical file.

  You cannot use a simple or multiple format logical file to bring together into one record format fields from separate physical files. Use a join logical file to accomplish this. A record read through the use of a record format in a simple or multiple format logical file can contain data from only one physical file, and a record written through the use of a record format in a logical file can be stored only in one physical file.

Distributed data management (DDM) files are allowed on the PFILE keyword only when the logical file is being created on a remote system.

Examples

The following examples show how to specify the PFILE keyword.

Example 1

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R LOGRCD1                   PFILE(PF1)
     A
```

In this example, LOGRCD1 can use fields only in PF1.

Example 2

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R LOGRCD2                   PFILE(PF1 PF2)
     A               :
     A               :
00020A          R LOGRCD3                   PFILE(PF1 PF2 PF3)
     A               :
     A               :
     A
```

In this example, LOGRCD2 must use fields common to PF1, and PF2, and LOGRCD3 must use fields common to PF1, PF2, and PF3.

Example 3

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R LOGRCD4                   PFILE(PF1)
     A               :
     A               :
00020A          R LOGRCD5                   PFILE(PF2)
     A               :
     A               :
00030A          R LOGRCD6                   PFILE(LIB1/PF6)
     A
```

In this example, LOGRCD4, LOGRCD5, and LOGRCD6 can have unique fields. LOGRCD6 specifies a qualified physical-file name.

RANGE (Range) keyword for physical and logical files

You can specify this keyword at the field level, the select- or omit-field level, or both.

The format of the keyword is:

```
RANGE(low-value high-value)
```

Example 1

The following example shows how to specify character and numeric strings for the RANGE keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD                    PFILE(PF1)
     A
00020A            FIELDA         1  0       RANGE(2 5)    1
00030A            FIELDB         1          RANGE('2' '5')
00040A            FIELDC
00050A          K FIELDD
00060A          S FIELDA                    RANGE(1   4)  2
     A
```

In this example, RANGE (1) is specified for FIELDA and FIELDB as a validity checking keyword for display files that refer to FIELDA and FIELDB. In the display file, RANGE requires that the workstation user type only 2, 3, 4, or 5 in FIELDA or FIELDB. FIELDA is a numeric field and FIELDB is a character field. The type of field you specify depends on the high-level language the program is written in.

RANGE (2) is specified as a select/omit keyword for FIELDA. Records from the physical file PF1 are retrieved through this logical file record format only if the value of FIELDA is 1, 2, 3, or 4.

Example 2

The following example uses hexadecimal character strings when specifying the RANGE keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RCD1                      PFILE(PF1)
00020A            CODEA
00030A            FLD1
00040A            FLD2
```

```
00050A          K FLD1
00060A          S CODEA                     RANGE(X'51' X'54')
     A
```

RANGE is specified as a select/omit keyword for CODEA, which is a 1-byte field. Records from physical file PF1 are retrieved through this record format only if the value of field CODEA is from hexadecimal 51 through hexadecimal 54.

Specifying RANGE at the field level

At the field level, this keyword specifies validity checking for the field that you are defining when it is referred to later during display file creation.

RANGE does not affect the physical or logical file you are defining. When you define an input-capable field in a display file, you can refer to the field that you are defining by specifying R in position 29 and the REF or REFFLD keyword. During display file creation, the operating system copies the RANGE keyword and other field attributes from the field in the physical or logical file into the field in the display file. You can override the RANGE keyword (as well as all other validity-checking keywords and the CHKMSGID keyword) by specifying any validity checking keyword for the field in the display file.

The rules for specifying this keyword in a physical or logical file are the same as the rules for a display file.

You cannot specify the RANGE keyword on a floating-point field (F in position 35) or a hexadecimal field (H in position 35). Do not specify the RANGE keyword on a date, time, or timestamp field (L, T, or Z in position 35).

Related information

Reference for display files (position 29)
RANGE (Range) keyword for display files

Specifying RANGE at the select/omit-field level

At the select/omit-field level, this keyword selects or omits records retrieved from the physical file(s) when your program sends an input operation using the record format in which the select/omit field is specified.

The following rules apply:

- If the field you are defining is a character field, you must specify character strings or hexadecimal character strings.

  Specify character strings enclosed in single quotation marks. See Example 1 in RANGE (Range) keyword for physical and logical files.

  Specify hexadecimal character strings as an X followed by a combination of the digits 0 through 9 and the letters A through F, enclosed in single quotation marks. The number of hexadecimal digits in single quotation marks must be exactly twice the specified length of the field. See Example 2 in RANGE (Range) keyword for physical and logical files.
- If you are defining a numeric field, you must specify a numeric value (digits 0 through 9 specified without single quotation marks). See Example 1 in RANGE (Range) keyword for physical and logical files.
- If you are defining a date field, specify a valid date in the same format specified on the DATFMT keyword and use the same separator as specified on the DATSEP keyword. For example, `RANGE('12/15/05' '12/31/05')` is the default value if *MDY is specified for DATFMT and '/' is specified for DATSEP.
- If you are defining a time field, specify a valid time in the same format specified on the TIMFMT keyword and use the same separator as specified on the TIMSEP keyword. For example, `RANGE('11.00.00' '12.00.00')` is the default value if *ISO is specified for TIMFMT. The default separator for *ISO is a period (.).
- If you are defining a timestamp field, you must specify the default value in the following format:

```
RANGE('YYYY-MM-DD-HH.MM.SS.UUUUUU' 'YYYY-MM-DD-HH.MM.SS.UUUUUU')
```

REF (Reference) keyword—physical files only

You can use this file-level keyword to specify the name of the file from which field descriptions are retrieved.

The format of the keyword is:

```
REF([library-name/]database-file-name [record-format-name])
```

REF supplies the field attributes from a previously defined record format. Specify the file name once in the REF keyword instead of on several REFFLD keywords if each field description refers to the same file. To refer to more than one file, use the REFFLD keyword. You can specify the REF keyword only once.

The database-file-name is a required parameter value for this keyword. The library-name and the record-format-name are optional.

If you do not specify the library-name, the library list (*LIBL) in effect at file creation time is used. Specify a record-format-name as a parameter value for this keyword if there is more than one record format. If you do not specify the record-format-name, each record format is searched sequentially. The first occurrence of the field name is used.

You can specify a distributed data management (DDM) file on this keyword.

When using a DDM file, the database-file-name and library-name are the DDM file and library names on the source system. The record-format-name is the record format name in the remote file on the target system.

Note: Interactive data definition utility (IDDU) files cannot be used as reference files.

Examples

The following example shows how to specify the REF keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                  REF(FILE1)
00020A          R RECORD
00030A            FLD1      R
     A
```

FLD1 has the same attributes as the first (or only) FLD1 in FILE1.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                  REF(LIB1/FILE1 RECORD2)
00020A          R RECORD
00030A            FLD1      R
     A
```

FLD1 has the same attributes as FLD1 in RECORD2 in FILE1 in LIB1.

Related reference

Reference for physical and logical files (position 29)
You use this position to specify reference for physical files only.

Related information

When to specify REF and REFFLD keywords for DDS files

REFACCPTH (Reference Access Path Definition) keyword—logical files only

You can use this file-level keyword to specify that the access path information for this logical file is to be copied from another physical or logical file.

The access path information includes key information, select and omit information, alternative collating sequence information, dynamic select information, and key sequencing information (specified in the FCFO, FIFO, LIFO, and UNIQUE keywords).

The format of the keyword is:

```
REFACCPTH([library-name/]database-file-name)
```

The name of the file defining the access path is the parameter value for the keyword.

The file containing the REFACCPTH keyword cannot contain key, select, or omit fields.

The record format(s) in the file you are defining can contain fewer or more fields than the record format(s) in the physical file on which this logical file is based.

If the file specified on the REFACCPTH keyword is a simple- or multiple-format logical file, the file containing the REFACCPTH keyword must have the same physical files specified in the same order on the PFILE keyword.

The REFACCPTH keyword is not allowed in join logical files. You can specify a join logical file as the parameter value on the REFACCPTH keyword only if all the following situations are true:

- The file you are creating is a simple logical file.
- The physical file specified on the PFILE keyword is the first file specified on the JFILE keyword in the join logical file.
- The join logical file has key fields specified and does not have select and omit fields specified.

You cannot specify a distributed data management (DDM) file on this keyword.

You cannot specify the REFACCPTH keyword with the DYNSLT, ALTSEQ, FCFO, FIFO, LIFO, or UNIQUE keywords.

Example

The following example shows how to specify the REFACCPTH keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00030A* ORDER HEADER LOGICAL FILE (ORDHDR11)
00040A                                  REFACCPTH(DSTLIB/ORDHDRL)
00050A          R ORDHDR                PFILE(ORDHDRP)
     A
```

REFFLD (Referenced Field) keyword—physical files only

You can use this field-level keyword to refer to a field under any of these conditions.

These conditions are:

- When the name of the referenced field is different from the name in positions 19 through 28
- When the name of the referenced field is the same as the name in positions 19 through 28, but the record format, file, or library of the referenced field is different from that specified with the REF keyword
- When the referenced field occurs in the same DDS source file as the referencing field

The format of the keyword is:

```
REFFLD([record-format-name/]referenced-field-name
[{*SRC | [library-name/]database-file-name}])
```

The referenced-field-name is required even if it is the same as the name of the field being defined. Use the record-format-name when the referenced file contains more than one record format. Use *SRC (rather than the database-file-name) when the field name being referred to is in the same DDS source file as the field being defined. *SRC is the default value when the database-file-name and the library-name are not specified.

Note: When you refer to a field in the same DDS source file, the field being referred to must precede the field being defined.

Specify the database-file-name (with its library-name, if necessary) to search a particular database file.

An R must be in position 29. Some keywords specified with the field being referred to are not included on the field being defined. For more information, see the Reference for physical and logical files (position 29) topic.

If you specify REF at the file level and REFFLD at the field level in the same DDS source file, the REFFLD specification is used. The search sequence depends on both the REF and REFFLD keywords.

You can specify a distributed data management (DDM) file on this keyword.

When using a DDM file, the database-file-name and library-name are the DDM file and library names on the source system. The referenced-field-name and the record-format-name are the field name and the record format name in the remote file on the target system.

Note: Interactive data definition utility (IDDU) files cannot be used as reference files.

Example

The following example shows how to code the REFFLD keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R FMAT1
00020A            ITEM           5
00030A            ITEM1     R               REFFLD(ITEM)
00040A            ITEM2     R               REFFLD(FMAT1/ITEM)
00050A            ITEM3     R               REFFLD(ITEM FILEX)
00060A            ITEM4     R               REFFLD(ITEM LIBY/FILEX)
00070A            ITEM5     R               REFFLD(FMAT1/ITEM LIBY/FILEX)
00080A            ITEM6     R               REFFLD(ITEM *SCR)
     A
```

The default for lines 00030 and 00040 is to search the DDS source file where they are specified because the REF keyword is not specified. In line 00080, the parameter *SRC explicitly specifies this source file.

Related reference

Reference for physical and logical files (position 29)
You use this position to specify reference for physical files only.

Related information

When to specify REF and REFFLD keywords for DDS files

REFSHIFT (Reference Shift) keyword for physical and logical files

You can use this field-level keyword to specify a keyboard shift for a field when the field is referred to in a display file or data file utility (DFU) operation.

The format of the keyword is:

```
REFSHIFT(keyboard-shift)
```

When defining an input-capable field in a display file, refer to the field that you are defining by specifying the letter R in position 29 and the REF or REFFLD keyword. At display file creation, the operating system copies the REFSHIFT keyword and other field attributes from the field in the logical file into the field in the display file. You can override the editing specified in the display or printer file by specifying new editing keywords. Specifying the DLTEDT keyword deletes all editing for the field.

The keyboard shift in the display file (position 35) becomes the parameter value specified on this keyword instead of the data type specified in the database file. When you refer to a field with the REFSHIFT keyword from a physical or logical file, the REFSHIFT keyword is copied into the new field. However, if the field attributes (such as data type) specified for the new field are not compatible with the keyboard shift specified on the REFSHIFT keyword, the keyword is ignored.

This keyword is valid for fields with data types A, S, B, or P. Choose any keyboard shift that is compatible with the data type as a parameter value. The following parameters apply to the data types:

- Character field (A): REFSHIFT(A | X | W | N | I | D | M)
- Numeric fields (S, B, P): REFSHIFT(S | Y | N | I | D)

Examples

The following example shows how to specify the REFSHIFT keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD
00020A            FIELDA         5          REFSHIFT(X)
00030A            FIELDN         4P         REFSHIFT(N)
     A
```

Fields FIELDA and FIELDN in the file (FILE1) have the REFSHIFT keyword specified as shown. The REFSHIFT keyword is used when the fields are referred to from a display file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A                                     REF(FILE1)
00020A          R RECORD
00030A            FIELDA    R           1  2
     A            FIELDN    R           2  2
     A
```

The display file references FILE1 (REF keyword). Fields FIELDA and FIELDN in this display file reference fields FIELDA and FIELDN in FILE1. When the REFSHIFT keyword is specified for the fields in FILE1, the keyboard shift specified with the REFSHIFT keyword is used in the display file, and the fields have the following attributes:

- FIELDA has keyboard shift X in position 35.
- FIELDN has keyboard shift N in position 35.

Related information

Reference for display files (position 29)

Data type/keyboard shift for display files (position 35)

RENAME (Rename) keyword—logical files only

You can use this field-level keyword to specify that a field name in the logical record format that you are defining is different from its corresponding physical-file field name.

The format of the keyword is:

```
RENAME(physical-file-field-name)
```

The name as it appears in the physical file record format is the parameter value for this keyword. One field in the physical file record format can be renamed to more than one field in the record format being described.

You can rename fields in situations similar to the following ones:

- You want to use programs that were written using a different name for the same field.
- You want to map one field in a physical file record format to two or more fields in a logical file record format.
- You are using a high-level language (such as RPG III) that does not permit two fields having different names to have only one data storage area. By specifying the RENAME keyword, you allow both fields to access the same data storage area.

If you specify the same physical field more than once in a record format in the logical file (that is, by using either RENAME or CONCAT), the sequence in which the fields are specified in the logical file is the sequence in which the data is moved to the physical file on an update or insert operation. Thus, the value in the last occurrence of the physical field is the value that is put in the physical record and is the value that is used for all keys built over that physical field. All previous values of the same field are written over and have no effect.

Examples

The following examples show how to specify the RENAME keyword.

In the following example, the QTYDUE field in the physical file (PF1) is renamed QTY in the logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00050A          R RCD1                      PFILE(PF1)
00060A            QTY                       RENAME(QTYDUE)
     A
```

In the following example, the renamed field in the logical file (QTY) is used as a key field.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00050A          R RCD2                      PFILE(PF2)
00060A             :
     A             :
00130A            QTY                       RENAME(QTYDUE)
00140A          K QTY
     A
```

SIGNED (Signed) keyword for physical and logical files

If this key field-level keyword is in effect, when sequencing the values associated with this numeric key field, the operating system considers the signs of the values (negative versus positive values).

This keyword has no parameters.

The following example shows six records with a zoned decimal key field:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
1	98	F9F8
2	00	F0F0
3	98-	F9D8
4	97	F9F7
5	20	F2F0
6	99	F9F9

By default (with no sequencing keywords specified and without the ALTSEQ keyword), the key field has the SIGNED attribute. The records are sequenced in the following order:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
3	98-	F9D8
2	00	F0F0
5	20	F2F0
4	97	F9F7
1	98	F9F8
6	99	F9F9

If both SIGNED and DESCEND are specified, the records are sequenced in this order:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
6	99	F9F9
1	98	F9F8
4	97	F9F7

Record	Numeric key field (zoned decimal)	Hexadecimal representation
5	20	F2F0
2	00	F0F0
3	98-	F9D8

This keyword is not valid for a character, date, time, timestamp, or hexadecimal data type field. You cannot use it with the ABSVAL, DIGIT, UNSIGNED, or ZONE keywords.

SIGNED (a key field-level keyword) causes ALTSEQ (a file-level keyword) to be ignored. If you specify SIGNED for a key field, NOALTSEQ is automatically in effect for that key field even if ALTSEQ is specified at the file level. This occurs, whether NOALTSEQ is specified.

Example

The following example shows how to specify the SIGNED keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD
00020A            FLDA           7S 2
00030A            FLDB
00040A          K FLDA                       SIGNED
     A
```

Related concepts

Key field name
When you specify K in position 17, the name specified in positions 19 through 28 is a key field name.

Related reference

DESCEND (Descend) keyword for physical and logical files
You can use this key field-level keyword to specify that the values of this character, hexadecimal, or numeric key field are retrieved in descending sequence.

SST (Substring) keyword—logical files only

You can use this field-level keyword to specify a character string that is a subset of an existing character, hexadecimal, zoned field, or graphic.

The format of the keyword is:

```
SST(field-name starting-position [length])
```

The field-name parameter specifies the name of the field from which the substring is taken. This field must be defined in the same logical file format before the SST field (which is the field you are defining) or it must exist in the physical file specified on the PFILE or JFILE keyword. To find the field, the system searches for a matching field name as follows:

1. First, the system searches the field names specified in positions 19 to 28 in the logical file format before the SST field.
2. If no matching field name is found in positions 19 to 28 in the logical file format, the system searches for the field name in the physical file specified on the PFILE or JFILE keyword, according to the following rules:
   - If the logical file is a simple or multiple format logical file, the field must exist in all files specified on the PFILE keyword.
   - If the logical file is a join logical file and the JREF keyword is specified on the SST field, the field must exist in the JFILE referred to by the JREF keyword.
   - If the logical file is a join logical file and the JREF keyword is not specified on the SST field, the field must exist in exactly one JFILE.

The substring begins at the starting position you specify on the SST keyword. Specify its length either as the third parameter on the keyword or on the field length (DDS positions 30 through 34). The starting position is a required parameter; the length is optional.

Note: Both the starting position and length values must be positive integer values and the defined substring must not be greater than the length of the field specified on the SST keyword.

The following rules apply:

- If the field on the SST keyword is binary character, the resulting field is binary character; if the field on the SST keyword is hexadecimal, the resulting field is hexadecimal; if the field on the SST keyword is DBCS-graphic, the resulting field is DBCS-graphic; otherwise, the resulting field is always character. If the data type is not specified in DDS, the result field's data type depends on the sub-stringed field as shown in the following table (the *Source field type* is the type of the physical file field or the logical file field defined earlier in the logical file source):

Source field type	**Logical file field becomes:**
A	A
H	H
S	A
G	G
Binary character	Binary character

- The use of the resulting field must be either input-only (I) or neither (N).
- The length of the resulting field is optional. You must specify either the field length or the length parameter on the keyword. If you specify both, they must be equal. If the field length is not specified, it is assigned the length parameter on the keyword.
- You cannot specify this keyword on the same field with the CONCAT, RENAME, or TRNTBL keywords.
- The field specified on this keyword cannot be defined with the CONCAT, TRNTBL, or SST keywords.

Examples

The following example shows how to specify the SST keyword on a join logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RECORD1                   JFILE(PF1 PF2)
     A          J                           JOIN(1 2)
     A                                      JFELD(CITY CITY)
     A            ADDRESS                   JREF(2)
     A            CITY              I       SST(ADDRESS 21 10)
     A                                      JREF(2)
     A            SYEAR             I       SST(SALESDATE 5)
     A            NAME                      JREF(1)
     A            CUSTNAME          I       SST(NAME 11 10) JREF(2)
     A          K SYEAR
     A
```

This example shows:

- CITY is a substring of ADDRESS from the logical format and is joined with CITY from PF1.
- CUSTNAME is a substring of NAME from PF2 because NAME in the logical file format has a different JREF.
- Because SYEAR is a key field, the unique field name SALESDATE must exist in PF1.
- The usage (position 38) for a field with the SST keyword must be I (input only). Because this is a join logical file, the usage default is I.

The following example shows how to specify the SST keyword on a simple or multiple format logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1                      PFILE(PFA)
```

```
     A            LASTNAME           I      SST(NAME 10 10)
     A          K LASTNAME
     A
```

The LASTNAME field is a substring of NAME from PFA. The usage I in position 38 must be specified for SST fields in simple or multiple format logical files.

Related reference

Length for physical and logical files (positions 30 through 34)
You use these positions to specify the length of a physical or logical file field.

TEXT (Text) keyword for physical and logical files

You can use this record- or field-level keyword to supply a text description (or comment) for the record format or field used for program documentation.

The format of the keyword is:

```
TEXT('description')
```

The text must be enclosed in single quotation marks. If the length of the text is greater than 50 positions, only the first 50 characters are used by the high-level language compiler.

Note: If the TEXT keyword is specified for a logical file and no fields are specified, the text keyword for the physical file is used (if specified).

Example

The following example shows how to specify the TEXT keyword at the record and field levels.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R CUSMST                    TEXT('Customer Master Record')
00020A            FLD1          3  0       TEXT('ORDER NUMBER FIELD')
     A
```

TIMFMT (Time Format) keyword for physical and logical files

You can use this field-level keyword to specify the format of a time field.

This keyword is valid for either time fields (data type T) or zoned fields (data type S) whose corresponding physical file fields are time fields (data type T).

The format of the keyword is:

```
TIMFMT(time-format)
```

The following table describes the valid time formats and their default separators.

Format name	Time format parameter	Time format and separator	Field length	Example
Hours:Minutes:Seconds	*HMS	hh:mm:ss	8	14:00:00
International Standards Organization	*ISO	hh.mm.ss	8	14.00.00
IBM USA Standard	*USA	hh:mm AM or hh:mm PM	8	2:00 pm
IBM European Standard	*EUR	hh.mm.ss	8	14.00.00

Format name	Time format parameter	Time format and separator	Field length	Example
Japanese Industrial Standard Christian Era	*JIS	hh:mm:ss	8	14:00:00

If you do not specify the TIMFMT keyword for a physical file, the default is *ISO.

If you do not specify the TIMFMT keyword for a logical file, the default is the time format from the physical file.

If you specify the time-format parameter value *ISO, *USA, *EUR, or *JIS, you cannot specify the TIMSEP keyword. These formats have a fixed separator.

The TIMFMT keyword overrides the job attribute for a time field. It does not change the system default.

Example

The following example shows how to specify the TIMFMT keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A
00020A          R RECORD
00030A            TIMFLD1         T         TIMFMT(*ISO)
00040A            TIMFLD2         T         TIMFMT(*USA)
     A
```

If the current time is 2 o'clock p.m., the system time format is hhmmss, and the system time separator is ':', TIMFLD1 contains 14.00.00. TIMFLD2 contains 2:00 PM.

TIMSEP (Time Separator) keyword for physical and logical files

You can use this field-level keyword to specify the separator character used for a time field. This keyword is valid only for time fields (data type T).

The format of the keyword is:

```
TIMSEP(*JOB | 'time-separator')
```

The time-separator parameter specifies the separator character that appears between the hour, minute, and second values. Valid values are a colon (:), a period (.), a comma (,), and blank ( ). The parameter must be enclosed in single quotation marks.

If you specify *JOB, the default is the job attribute.

For physical files, if you do not specify the TIMSEP keyword, the default is the job attribute.

For logical files, if you do not specify the TIMSEP keyword, the default is the separator character from the physical file. If you did not specify the TIMSEP keyword for the physical file field (*ISO, *USA, *EUR, or *JIS was specified on the TIMFMT keyword), the default is the job attribute.

If you specify *ISO, *USA, *EUR, or *JIS time format on the TIMFMT keyword, you cannot specify the TIMSEP keyword. These formats have a fixed separator.

If the DFT keyword is not specified, the default value is the current time.

The TIMSEP keyword overrides the job attribute for a time field. It does not change the system default.

Example

The following example shows how to specify the TIMSEP keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A
00020A          R RECORD
```

```
00030A            TIMFLD1         T         TIMSEP(' ')
00040A            TIMFLD2         T         TIMSEP('.')
```

If the current time is 2 o'clock p.m., the system time format is hhmmss, and the system time separator is ':', TIMFLD1 contains 14 00 00. TIMFLD2 contains 14.00.00.

TRNTBL (Translation Table) keyword—logical files only

You can use this field-level keyword to specify the name of a translation table to be used when the system passes this field between the physical file on the PFILE or JFILE keyword and your program.

The field must be a character field and its length cannot be redefined in the logical file. If the TRNTBL keyword is specified with the CONCAT keyword, the fields specified on the CONCAT keyword must all be character fields.

The format of the keyword is:

```
TRNTBL([library-name/]translation-table-name)
```

The translation-table-name is a required parameter value; the library-name is optional. If you do not specify the library-name, the operating system uses the library list (*LIBL) that is in effect at file creation time.

The translation-table-name cannot name an ICU table.

This keyword is valid only for character fields that are input-only (I specified in position 38) or neither (N specified in position 38) fields.

You cannot specify the TRNTBL keyword on a hexadecimal field (H in position 35). Do not specify the TRNTBL keyword on a date, time, or timestamp field (L, T, or Z in position 35).

You can specify as many as 99 different translation tables for different fields in the same logical file.

Translation occurs when the field is read from the physical file. Therefore, all functions specified in the logical file (such as key field sequencing, select/omit processing, and joining of records) depend on the translated version of the data.

The TRNTBL keyword changes the data in the records returned from the logical file. The ALTSEQ keyword changes only the order of the records returned from the logical file.

The TRNTBL keyword is similar to the CHRID keyword but differs from the latter as follows:

- The TRNTBL keyword names the translation table to be used; the CHRID keyword does not.
- The TRNTBL keyword changes data on input to the program when your program is reading a logical file. The CHRID keyword changes the data for display or printing on a specific device. Use the TRNTBL keyword when your program will use the changed data (for example, in an IF-THEN-ELSE statement or a COBOL SORT statement). If your program is handling data that comes only from a logical file, you do not need to specify the CHRID keyword in display or printer files used by the program.

The TRNTBL keyword is not valid when you specify FILETYPE(*SRC) on the Create Logical File (CRTLF) command.

Notes:

1. When you use the TRNTBL keyword, the length of the field in the logical file must be the same as the length of the corresponding field in the physical file.
2. At file creation time, you must have use authority to the translation table. The translation table is created using the Create Table (CRTTBL) command.
3. The translation table specified in the TRNTBL keyword is referred to only when the logical file is created. Therefore, a change to a translation table does not affect the logical file until the logical file is re-created.

Example

The following example shows how to specify the TRNTBL keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                 PFILE(PF1)
00020A            CHAR1                I  TRNTBL(LIB1/TBL1)
00030A            CHAR2              A  I  TRNTBL(LIB2/TBL2)
00040A            NUM1
00050A            NUM2
     A
```

Field CHAR1 is translated using table TBL1 in library LIB1. Field CHAR2 is translated using table TBL2 in library LIB2. Field CHAR2 was redefined in the logical file as a character field (A in position 35) to allow the TRNTBL keyword to be specified. Fields NUM1 and NUM2 are numeric fields in the physical file PF1 and cannot have the TRNTBL keyword specified for them.

UNIQUE (Unique) keyword for physical and logical files

You can use this file-level keyword to specify that records with duplicate key values are not allowed within a member of this physical or logical file.

You can specify whether null key values are to be considered as duplicates using the parameter. Any insertions or additions of new records, or updates to existing records, which might result in a duplicate key, are rejected. The application program issuing the write or the update operation receives an error message. When a workstation user is using data file utility (DFU), a message is displayed at the workstation. A copy file command that copies records with duplicate keys in this file is not completed.

The format of this keyword is:

```
UNIQUE[(*INCNULL | *EXCNULL)]
```

The parameter is optional. When specified, it determines whether null key values cause duplicates. *INCNULL is the default and indicates to include null values when determining duplicates. *EXCNULL, when specified, indicates to exclude null values when determining duplicates.

When a logical file based on a physical file has the UNIQUE keyword, the physical file member or members cannot have duplicate key values.

When you specify the UNIQUE keyword for a physical or logical file, you must specify the MAINT(*IMMED) parameter value on the Create Physical File (CRTPF) or Create Logical File (CRTLF) command that creates the file. This means that the access path is maintained immediately when changes are made.

If you do not specify the UNIQUE keyword, records with duplicate key values are sequenced in the order you specify. If you specify the FIFO keyword, they are sequenced in first-in first-out order. If you specify the LIFO keyword, they are sequenced in last-in first-out order. If you specify the FCFO keyword, they are sequenced in first-changed first-out order. If you do not specify FIFO, LIFO or FCFO, the order in which the records are sequenced is not guaranteed.

You cannot specify the UNIQUE keyword with the FIFO, LIFO, FCFO, or REFACCPTH keywords.

Example

The following example shows how to specify the UNIQUE keyword for a logical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A*
00020A* SAMPLE LOGICAL FILE (CUSMSTL)
00030A*
00040A                                    UNIQUE
00050A          R CUSREC                  PFILE(CUSMSTP)
00060A                                    TEXT('Logical File Master Record')
00070A            CUST
00080A            NAME
00090A            ADDR
00100A          K CUST
     A
```

UNSIGNED (Unsigned) keyword for physical and logical files

You can use this key field-level keyword to specify that numeric fields are sequenced as a string of unsigned binary data. The default values of character, date, time, timestamp, and hexadecimal fields are set to unsigned values.

This keyword has no parameters.

UNSIGNED is valid on key fields in physical or logical files regardless of the data type of the key field. The UNSIGNED keyword is not allowed with the SIGNED and ABSVAL keywords.

The UNSIGNED keyword will be the default in the following situations:

- When you specify ALTSEQ at the file level for a zoned key field
- When you specify ZONE or DIGIT for a zoned key field
- For all character and hexadecimal fields

Note: You can specify UNSIGNED for floating point fields, but the results cannot be predicted.

The following figure shows six records with a zoned decimal key field:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
1	98	F9F8
2	00	F0F0
3	98-	F9D8
4	97	F9F7
5	20	F2F0
6	99	F9F9

If you specify UNSIGNED, the records are sequenced in this order:

Record	Numeric key field (zoned decimal)	Hexadecimal representation
2	00	F0F0
5	20	F2F0
3	98-	F9D8
4	97	F9F7
1	98	F9F8
6	99	F9F9

Example

The following example shows how to specify the UNSIGNED keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A         R RECORDA
00020A           FLDA           7S 2
00030A           FLDB           5
00040A         K FLDA                      UNSIGNED
     A
```

VALUES (Values) keyword for physical and logical files

You can specify this keyword at the field level, the select/omit-field level, or both.

The format of the keyword is:

```
VALUES(value-1 [value-2...[value-100]])
```

Examples

The following examples show how to specify the VALUES keyword.

Example 1

The following example uses character and numeric values.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1                   PFILE(PF1)
00020A            FIELDA         1  0       VALUES(1 6 9)        1
00030A            FIELDB         1          VALUES('A' 'B' 'C')  1
00040A          K FIELDA
00050A          S FIELDB                    VALUES('A' 'B')      2
00060A          S FIELDA                    VALUES(1 6)          2
     A
```

1
VALUES is specified for FIELDA and FIELDB as a validity checking keyword for display files that refer to FIELDA and FIELDB.

2
VALUES is also specified for FIELDA and FIELDB as a select/omit keyword. Records from the physical file PF1 are retrieved through this logical file record format depending on the values of the following fields:

- FIELDB: Records are selected only when FIELDB equals A or B.
- FIELDA: Records not already selected are selected when FIELDA equals 1 or 6.

Example 2

The following example uses hexadecimal values.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RCD1                      PFILE(PF1)
00020A            CODEA
00030A            FLD1
00040A            FLD2
00050A          K FLD1
00060A          S CODEA                     VALUES(X'51' X'54' X'AE')
     A
```

VALUES is specified as a select/omit keyword for CODEA (which is a 1-byte field). Records from physical file PF1 are retrieved through this record format only if the value of field CODEA is hexadecimal 51, hexadecimal 54, or hexadecimal AE.

Specifying VALUES at the field level

At the field level, this keyword specifies validity checking for the field that you are defining when the field is referred to later during display file creation.

VALUES does not affect the physical or logical file that you are defining. When you define an input-capable field in a display file, you can refer to the field that you are defining by specifying R in position 29 and the REF or REFFLD keyword. During display file creation, the operating system copies the VALUES keyword and other field attributes from the field in the physical or logical file into the field in the display file. You can override the VALUES keyword (as well as all other validity-checking keywords and the CHKMSGID keyword) by specifying any validity checking keyword for the field in the display file.

The rules for specifying this keyword in a physical or logical file are the same as the rules for a display file.

You cannot specify the VALUES keyword on a floating-point field (F in position 35) or a hexadecimal field (H in position 35). Do not specify the VALUES keyword on a date, time, or timestamp field (L, T, or Z in position 35).

Related information

Reference for display files (position 29)
VALUES (Values) keyword for display files

Specifying VALUES at the select/omit-field level

At the select/omit-field level, this keyword selects or omits records retrieved from the physical files when your program sends an input operation using the record format in which the select/omit field is specified.

The following rules apply:

- If the field you are defining is a character field, you must specify character strings or hexadecimal character strings.

  Specify character strings enclosed in single quotation marks. See Example 1 in VALUES (Values) keyword for physical and logical files.

  Specify hexadecimal character strings as an X followed by a combination of the digits 0 through 9 and the letters A through F, enclosed in single quotation marks. The number of hexadecimal digits in single quotation marks must be exactly twice the specified length of the field. See Example 2 in VALUES (Values) keyword for physical and logical files.
- If the field you are defining is a numeric field, you must specify a numeric value (digits 0 through 9 specified without single quotation marks). See Example 1 in VALUES (Values) keyword for physical and logical files.
- If you are defining a date field, specify a valid date in the same format specified on the DATFMT keyword and use the same separator as specified on the DATSEP keyword. For example, `VALUES('12/15/05' '12/31/05')` is the default value if *MDY is specified for DATFMT and '/' is specified for DATSEP.
- If you are defining a time field, specify a valid time in the same format specified on the TIMFMT keyword and use the same separator as specified on the TIMSEP keyword. For example, `VALUES('11.00.00' '12.00.00')` is the default value if *ISO is specified for TIMFMT. The default separator for *ISO is a period (.).
- If you are defining a timestamp field, you must specify the default value in the following format:

```
VALUES('YYYY-MM-DD-HH.MM.SS.UUUUUU' 'YYYY-MM-DD-HH.MM.SS.UUUUUU')
```

VARLEN (Variable-Length Field) keyword for physical and logical files

You can use this field-level keyword to define this field as a variable-length field.

Variable-length fields are useful for improving storage when the data for the field typically fits within a certain length, but can occasionally be longer. Specify the maximum length of the field in positions 30 to 34. You can specify the allocated length (or typical length) in the parameter.

The format of the keyword is:

```
VARLEN[(allocated-length)]
```

The allocated-length parameter is optional. Use it to specify the number of bytes (two byte characters in the case of graphic fields) allocated for the field in the fixed portion of the file. If you do not specify the allocated-length parameter, the data for this field is stored in the variable length portion of the file.

Valid values for the allocated-length parameter are 1 to the maximum length of the field specified in positions 30 to 34.

The VARLEN keyword has no parameters for a logical file.

The VARLEN keyword is valid only on character fields and graphic fields.

When you specify the VARLEN keyword, the maximum length you can specify in positions 30 to 34 is 32 740 (32 739 if the field allows the null value). If the field is a graphic field, the maximum length you can specify is 16 370.

If you specify the DFT keyword for a variable-length field, the length of the default value must be less than or equal to the allocated length for the field. If the default value is longer than the allocated length, an error message is issued when the file is created.

If you specify a hexadecimal value as the default value for a variable-length field, the number of hexadecimal characters must be equal to two times the allocated length for the field.

The DFT keyword is not allowed on the same field as a VARLEN keyword unless you specify a value for the allocated-length parameter.

Do not specify the VARLEN keyword on a date, time, or timestamp field (L, T, or Z in position 35).

Example

The following example shows how to specify the VARLEN keyword for a physical file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          R RECORD1
00020A            FIELD1       100A         VARLEN(30)
00030A            FIELD2       200A         VARLEN
     A
```

FIELD1 is defined as a variable-length field with a maximum length of 100 and an allocated length of 30. FIELD2 is defined as a variable-length field with a maximum length of 200 and no allocated length.

ZONE (Zone) keyword for physical and logical files

You can use this key field-level keyword to specify that only the zone portion (farthest left 4 bits) of each byte of the key field is used when the operating system is constructing a value associated with this key field. The digit portion is filled with zeros.

This keyword has no parameters.

This keyword is applied against the entire key field (not just a position within the field) and is valid only for character, hexadecimal, or zoned decimal type fields.

ZONE is not allowed with the ABSVAL, SIGNED, or DIGIT keywords.

If you specify ZONE for a key field, the value of the field is treated as a string of unsigned binary data rather than signed (which is the default for zoned decimal fields).

Example

The following example shows how to specify the ZONE keyword.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
00010A          K CODE                       ZONE
     A
```

If CODE is a 1-byte field, the values of the field for three different records might be as follows:

Values	Hexadecimal	Digits used for key
A	C1	C
B	C2	C
E	C5	C

Unicode considerations for database files

Unicode is a universal encoding scheme for written characters and text that enables the exchange of data internationally.

This topic discusses how to specify DDS positions 30 through 37 and positions 45 through 80 for describing database files. Positions not mentioned have no special considerations for Unicode.

A Unicode field can contain all types of characters used on the IBM i platform, including double-byte character set (DBCS) characters. Unicode data is composed of *code units*, which represent the minimal byte combination that can represent a unit of text.

These transformation formats (encoding forms) of Unicode are supported with physical and logical file DDS:

- **UTF-8** is an 8-bit encoding form designed for ease of use with existing ASCII-based systems. UTF-8 data is stored in character data types. The CCSID value for data in UTF-8 format is 1208.

  A UTF-8 code unit is 1 byte in length. A UTF-8 character can be 1, 2, 3, or 4 code units in length. A UTF-8 data string can contain any character, including surrogates and combining characters.
- **UTF-16** is a 16-bit encoding form designed to provide code values for over a million characters, and a superset of UCS-2. UTF-16 data is stored in graphic data types. The CCSID value for data in UTF-16 format is 1200.

  A UTF-16 code unit is 2 bytes in length. A UTF-16 character can be 1 or 2 code units (2 or 4 bytes) in length. A UTF-16 data string can contain any character, including UTF-16 surrogates and combining characters.
- **UCS-2** is the Universal Character Set coded in 2 octets, which means that characters are represented in 16-bits per character. UCS-2 data is stored in graphic data types. The CCSID value for data in UCS-2 format is 13488.

  UCS-2 is a subset of UTF-16, and can no longer support all of the characters defined by Unicode. UCS-2 is identical to UTF-16, except that UTF-16 also supports combining characters and surrogates. If you do not need support for combining characters and surrogates, then you can choose to use the UCS-2 type, because there is more database functionality available for it.

Note: In this topic, references to UTF-16 imply UCS-2 as well.

Unicode considerations for database files: Length (positions 30 through 34)

The length of a field containing UTF-16 data can range from 1 through 16 383 code units. The length of a field containing UTF-8 data can range from 1 through 32 766 code units.

When determining the program length of a field containing Unicode data, consider the following rules:

- Each UTF-16 code unit is 2 bytes long.
- The length of the field is specified in the number of UTF-16 code units. For example, a field containing 3 UTF-16 code units has 6 bytes of data.
- Each UTF-8 code unit is 1 byte long. A UTF-8 character can be 1, 2, 3, or 4 code units in length.
- After the conversion between Unicode data and EBCDIC, the resulting data should be equal to, longer, or shorter than the original length of the data. For example, one UTF-16 code unit is composed of 2 bytes of data. That character might convert to one single-byte character set (SBCS) character composed of 1 byte of data, 11 graphic double-byte character set (DBCS) characters composed of 2 bytes of data, or one bracketed DBCS character composed of 4 bytes of data. Therefore, it is suggested that when a Unicode field (in the physical file) is converted to a field with a different type in the logical file, the field in the logical file be defined with the VARLEN keyword. The length of the logical file field should be defined large enough to hold the maximum size that the Unicode field can be converted to. This accounts for the expansion that can occur.

On a logical file, if the length is not specified, and a UTF-16 to EBCDIC conversion will be taking place, the length of the corresponding physical file field will be taken, except in the following case:

- If the physical file field is UTF-16 capable and the logical file field has a data type of O, then the length of the logical file field will be 2 times the field size of the physical file field.

Unicode considerations for database files: Data type (position 35)

The valid data types for Unicode data are the G (Graphic) data type and the A (Character) type.

G (Graphic)
Type G, in combination with the CCSID keyword, specifies that this field contains UTF-16 data.

Normally, by specifying G, the field contains graphic-DBCS EBCDIC data. In combination with the CCSID keyword, the field contains UTF-16 data. When conversion is necessary between corresponding fields in a physical and logical file, data will be mapped between the characters of the UTF-16 CCSID and the CCSID of the corresponding field.

A (Character)
Type A, in combination with the CCSID keyword (with 1208), specifies that this field contains UTF-8 data.

Normally, by specifying A, the field contains EBCDIC data. In combination with the CCSID keyword, the field contains UTF-8 data. When conversion is necessary between corresponding fields in a physical and logical file, data will be mapped between the characters of the UTF-8 CCSID and the CCSID of the corresponding field.

Related reference
Data type for physical and logical files (position 35)
For a physical file, you use this position to specify the data type of the field within the database. You specify data type in a logical file only to override or change the data type of the corresponding field in the physical file on which this logical file is based.

Unicode considerations for database files: Decimal positions (positions 36 and 37)

Leave these positions blank when using Unicode data.

Unicode considerations for database files: Keyword considerations (positions 45 through 80)

This information discusses how Unicode data is used with some DDS keywords.

The CCSID keyword is used to enable an A-type or G-type field to contain Unicode data.

The CCSID parameter must have a CCSID using a Unicode encoding scheme. This keyword is enabled for both physical and logical files.

For logical files, the following characteristics must be true before the CCSID keyword is allowed on a logical file field.

- If the specified value on the logical file CCSID keyword uses Unicode encoding schemes, the field data type must be A for UTF-8 or G for UCS-2/UTF-16. Also, the corresponding physical file field must be of types A, G, or O. If the CCSID keyword is specified on the physical file field, it must contain a value other than 65 535.
- If the specified value on the logical file CCSID keyword does not use Unicode encoding schemes, the field data type must be A, O, or G. Also, the corresponding physical file field must be a Unicode field. The CCSID keyword specified on the logical file field must contain a value other than 65 535.

The DFT keyword can contain SBCS, bracketed-DBCS, or bracketed-DBCS-graphic character strings when specified on a Unicode-capable field.

You can use the COMP keyword only to compare data in another Unicode-capable field. Two equal length UTF-8 data strings can be compared to each other using their hex values regardless of character boundaries.

You can specify a character literal on a select or omit field that is tagged with a Unicode CCSID on the COMP, RANGE, and VALUES keywords. The maximum length of the literal is equal to the number of Unicode code units that is defined in positions 30 to 34 of the DDS specification.

The VARLEN keyword can be used on Unicode fields.

Logical files can have UTF-8 and UTF-16 data keys.

Concatenation of Unicode fields

Unicode fields can be used in the CONCAT keyword. The following rules apply:

- The parameters of the CONCAT keyword can be UCS-2 graphic fields, UTF-16 fields, or a mix of each type. No other field types can be concatenated with UCS-2 or UTF-16 fields.
  - The concatenation result is UTF-16 if one of the parameters is UTF-16.
  - Otherwise, the result is UCS-2.
- A UTF-8 field can be concatenated only with other UTF-8 fields. No other field type is allowed to be concatenated with a UTF-8 field. The concatenation result is UTF-8.
- The resulting field must be an input-only field; use **I** in position 38 of the DDS source statement.

Join logical file support

A UTF-8 field can be joined to another UTF-8 field in a join logical file. UTF-16 fields can be joined. To join a UTF-8 or UTF-16 field to a CHAR field, for example, the UTF-8, UTF-16, or CHAR field must be redefined in the logical file to the same type field.

Select/Omit fields in logical files

UTF-8 and UTF-16 fields are allowed as select and omit field specifications.

DBCS considerations for database files

Be aware of these double-byte character set (DBCS) considerations for the positional entries and keyword entries for physical and logical files, along with general considerations for database files.

The functions described in these topics are supported on both DBCS and non-DBCS systems.

Related information

General considerations for using DBCS text with DDS files

Positional entry considerations for database files that use DBCS

Be aware of these double-byte character set (DBCS) considerations for the length, data type, and decimal positional entries on database files. Positional entries that are not mentioned have no special considerations for DBCS.

Length (positions 30 through 34)

The length of a field containing bracketed-DBCS data can range from 4 through 32 766 bytes (4 through 32 740 bytes if the field is variable length). The length of a DBCS-graphic field can range from 1 through 16 383 characters (1 through 16 370 characters if the field is variable length).

When determining the length of a field containing DBCS data, consider the following rules:

- Each DBCS character is 2 bytes long.
- For DBCS-graphic fields, the length of the field is specified in the number of DBCS characters.
- Include both shift-control characters in the length of the field for fields with a data type of J, E, or O. Together, these characters are 2 bytes long.
- Fields specified with the J or E data type must have an even length.

For example, a bracketed-DBCS field that contains up to 3 DBCS characters, 1 shift-in character, and 1 shift-out character, has 8 bytes of data:

```
(3 characters x 2 bytes) + (shift-out + shift-in) = 8
```

A DBCS-graphic field that contains up to 3 DBCS characters has 6 bytes of data:

```
(3 characters x 2 bytes) = 6
```

Data type (position 35)

You can use any of the four DBCS data types: J (Only), E (Either), O (Open) and G (Graphic).

J (Only)
Fields can contain only DBCS data.

E (Either)
Fields can contain either DBCS or alphanumeric data.

O (Open)
Fields can contain both DBCS and alphanumeric data. Distinguish DBCS data from alphanumeric data with shift-control characters.

G (Graphic)
Fields can contain only DBCS data with no shift-control characters.

Related reference

Data type for physical and logical files (position 35)
For a physical file, you use this position to specify the data type of the field within the database. You specify data type in a logical file only to override or change the data type of the corresponding field in the physical file on which this logical file is based.

Decimal (positions 36 and 37)

Leave these positions blank when using DBCS data.

Keyword considerations for database files that use DBCS

You should not specify DDS keywords to be used with numeric data for fields containing double-byte character set (DBCS) data. The system treats DBCS data the same as character data, and, therefore, cannot perform arithmetic operations on it.

Do not use the following DDS keywords with DBCS data fields (the data type specified in position 35 is O, J, E, or G):

ABSVAL	CHECK(VNE)	REFSHIFT
ALTSEQ	DATFMT	SIGNED
CHECK(M10)	DATSEP	SST
CHECK(M10F)	DIGIT	TIMFMT
CHECK(M11)	EDTCDE	TIMSEP
CHECK(M11F)	EDTWRD	TRNTBL
CHECK(VN)	FLTPCN	ZONE

Notes:

1. The SST keyword is allowed on fields with a data type of G.
2. The REFSHIFT keyword is allowed on fields with a data type of O, J, or E.

Related reference

Keyword entries for physical and logical files (positions 45 through 80)

Keyword entries are typed in positions 45 through 80 (functions).

CONCAT (Concatenate) keyword

Using this field-level keyword, you can combine two or more fields from the physical-file record format into one field in the logical-file record format you are describing.

The name of this concatenated field must appear in positions 19 through 28. Specify the physical file field names in the order in which you want them to be concatenated, and separate them by blanks.

The following rules and restrictions apply:

- The operating system assigns the length of the concatenated field as the sum of the lengths (digits and characters) of the fields that are included in the concatenation.

  Note: For fields with data type J, the shift-out and shift-in pairs between the concatenated fields are removed from the resulting field. If the resulting data type is hexadecimal, the shift-out and shift-in pairs are eliminated for DBCS fields that precede the first hexadecimal fields.
- A DBCS-graphic field can be concatenated only with another graphic data type field.
- The operating system assigns the data type based on the data types of the fields that are being concatenated. When bracketed-DBCS fields are included in a concatenation, the general rules are:
  - If the concatenation contains one or more hexadecimal (H) fields, the resulting data type is hexadecimal (H).
  - If all fields in the concatenation are DBCS-only (J), the resulting data type is DBCS-only (J).
  - If the concatenation contains one or more DBCS (O, E, J) fields, but no hexadecimal fields, the resulting data type is DBCS-open (O).
- The operating system assigns the field to be fixed length or variable length based on the fields that are being concatenated. The general rules are:
  - Concatenation of a variable length field to either a fixed length field or another variable length field results in a variable length field.
  - Concatenation of a fixed length field to a fixed length field results in a fixed length field unless the VARLEN keyword is also specified on the same field as the CONCAT keyword.
- The maximum length of a concatenated field varies depending on the data type of the concatenated field and the length of the fields being concatenated. If the concatenated field is zoned decimal (S), its total length cannot exceed 63 bytes; if it is character (A) or DBCS(O, J), its total length cannot exceed 32 766 bytes. If the concatenated field is variable length, its total length cannot exceed 32 740 bytes (32 739 if the field also allows the null value).

  If the concatenated field is a DBCS-graphic (G) field, its total length cannot exceed 16 383 characters. If the concatenated field is variable length, its total length cannot exceed 16 370 characters.
- In join logical files, the fields to be concatenated must be from the same physical file. The first field specified on the CONCAT keyword identifies which physical file is used. The first field must, therefore, be unique among the physical files on which the logical file is based, or you must also specify the JREF keyword to specify which physical file to use.
- When one or more of the fields being concatenated are DBCS fields, none of the fields on the CONCAT keyword can be specified as a key, select, or omit field unless the field name is also specified in positions 19 through 28 or on a RENAME or CONCAT keyword specified before the DBCS concatenation.
- The usage of a concatenated field must be I (input only).
- REFSHIFT cannot be specified on a concatenated field that has been assigned a data type of O or J.

Example

The following example shows how to specify the CONCAT keyword on the DDS form.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RECORD 1                PFILE(PF1)
     A            FLD1              I     CONCAT(PFLD1 PFLD2)
```

```
     A            FLD2               I      CONCAT(PFLD1 PFLD2 PFLD3)
     A            FLD3               I      CONCAT(PFLD4 PFLD5)
     A
```

In the example, if the fields from PF1 are:

- PFLD1 with data type J
- PFLD2 with data type J
- PFLD3 with data type E
- PFLD4 and PFLD5 with data type G

Then the resulting fields are:

- FLD1 with data type J
- FLD2 with data type O
- FLD3 with data type G

Related reference

CONCAT (Concatenate) keyword—logical files only
You can use this field-level keyword to combine two or more fields from the physical-file record format into one field in the logical-file record format you are defining. The name of this concatenated field must appear in positions 19 through 28.

Additional considerations for describing database files that contain DBCS data

Be aware of these considerations when describing a database file that contains double-byte character set (DBCS) fields.

- If you describe DBCS fields in the DDS, the system treats the file as a DBCS file. You do not need to specify IGCDTA(*YES) on the file creation command to identify the file as DBCS.
- The data type of a field in a physical file can be changed as follows when you refer to that field in a logical file:

Physical file data type	Logical file data type
J	J, O, E, H, G
O	O, H
E	O, E, H
A	A, O, E, H
H	J, O, E, A, H
G	G, O, J, E

  Note: When the physical file data type is character (A) or hexadecimal (H), and the logical file data type is DBCS-only (J) or DBCS-either (E), the physical file field length (columns 30 through 34) must be an even number greater than or equal to 4.
- DDS treats DBCS key fields as character fields (the data type specified in position 35 is O).
- DDS uses the EBCDIC collating sequence to sort DBCS data.
- Any key field sequencing keywords that can be used with character fields can be used with DBCS fields, except the following keywords:

  ALTSEQ
  DIGIT
  ZONE
- Use bracketed-DBCS data anywhere comments and character strings are allowed.

- Any bracketed-DBCS field except a field with data type J can be compared with a character field (data type A).
- A DBCS-graphic field can be compared only with another graphic field.
- The following validity checking keywords can be specified on DBCS fields:

  COMP
  RANGE
  VALUES
- When specifying the VARLEN keyword in a physical file, the minimum allowed length for the allocated length is 4 for a bracketed-DBCS field. The minimum allowed length for the allocated length is 1 for a DBCS-graphic field.

Related information

DBCS character strings

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

© Copyright IBM Corp. 1999, 2013

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Programming interface information

This SQL call level interface publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Index

Special Characters

*NONE key field 9, 13

A

Absolute Value (ABSVAL) keyword 31
ABSVAL (Absolute Value) keyword 31
ALIAS (Alternative Name) keyword 32
ALL (All) keyword 32
Allow Null (ALWNULL) keyword 34
Alternative Collating Sequence (ALTSEQ) keyword 33
Alternative Name (ALIAS) keyword 32
ALTSEQ (Alternative Collating Sequence) keyword 33
ALWNULL (Allow Null) keyword 34

C

CCSID (Coded Character Set Identifier) keyword 34
Change Logical File (CHGLF) command 49
CHECK (Check) keyword 36
Check Message Identifier (CHKMSGID) keyword 37
CHGLF (Change Logical File) command 49
CHKMSGID (Check Message Identifier) keyword 37
CMP (Comparison) keyword 37
Coded Character Set Identifier (CCSID) keyword 34
COLHDG (Column Heading) keyword 37
Column Heading (COLHDG) keyword 37
comment 5
COMP (Comparison) keyword 38
Comparison (CMP) keyword 37
CONCAT (Concatenate) keyword 41, 86
conditioning 5
converting between zoned decimal and hexadecimal data types 23, 26
converting data types
    when substringing fields 23, 27
converting data types when concatenating fields 23, 26
converting floating point
    to binary 23, 26
    to packed decimal 23, 26
    to zoned decimal 23, 26
Copy File (CPYF) command 47
copying
    file 47
CPYF (Copy File) command 47
Create Logical File (CRTLF) command 49
CRTLF (Create Logical File) command 49

D

data type 23
data types
    converting between zoned decimal and hexadecimal data types 23, 26
data types *(continued)*
    converting floating point to packed, zoned, and binary data types 23, 26
    converting in DDS files 23, 26
    converting when concatenating fields 23, 26
    converting when substringing fields 23, 27
Date Format (DATFMT) keyword 43
Date Separator (DATSEP) keyword 45
DATFMT (Date Format) keyword 43
DATSEP (Date Separator) keyword 45
DBCS considerations for database files 84
DDS file considerations
    Unicode 82
decimal positions 27
Default (DFT) keyword 46
defining a physical file 1
DESCEND (Descend) keyword 46
DFT (Default) keyword 46
DIGIT (Digit) keyword 48
Dynamic Select (DYNSLT) keyword 49
DYNSLT (Dynamic Select) keyword 49

E

Edit Code (EDTCDE) keyword 51
EDTCDE (Edit Code) keyword 51

F

FCFO (First-Changed First-Out) keyword 52
field name 8
FIFO (First-In First-Out) keyword 53
First-Changed First-Out (FCFO) keyword 52
First-In First-Out (FIFO) keyword 53
Floating-Point Precision (FLTPCN) keyword 53
FLTPCN (Floating-Point Precision) keyword 53
form type 5
FORMAT (Format) keyword 54

I

Initialize Physical File Member (INZPFM) command 47
initializing
    physical file member 47
INZPFM (Initialize Physical File Member) command 47

J

JDFTVAL (Join Default Values) keyword 54
JDUPSEQ (Join Duplicate Sequence) keyword 55
JFILE (Joined Files) keyword 57
JFLD (Joined Fields) keyword 58
JOIN (Join) keyword 60
Join Default Values (JDFTVAL) keyword 54
Join Duplicate Sequence (JDUPSEQ) keyword 55
Join Reference (JREF) keyword 62

Joined Fields (JFLD) keyword 58
Joined Files (JFILE) keyword 57
JREF (Join Reference) keyword 62

K

key field name 8
keyword entries for physical and logical files 30

L

Last-In First-Out (LIFO) keyword 63
length 21
LIFO (Last-In First-Out) keyword 63
location 30
logical files
  *NONE key field 9, 13
  comment 5
  conditioning 5
  data type 23
  DBCS considerations 84
  decimal positions 27
  field name 8
  form type 5
  key field name 8
  keyword entries 30
  length 21
  location 30
  multiple format 2
  multiple record formats 12
  name 6
  name type 5
  positional entries 4
  record format for simple and multiple format 7
  record format in join logical files 7
  record format name 6
  reference 20
  reserved 6
  select/omit field name 18
  sequence number 5
  simple 2
  specification type 5
  Unicode 82
  usage 28

M

multiple format logical files 2

N

name 6
name type 5
No Alternative Collating Sequence (NOALTSEQ) keyword 63
NOALTSEQ (No Alternative Collating Sequence) keyword 63

P

PFILE (Physical File) keyword 64
Physical File (PFILE) keyword 64
physical files
  comment 5
  conditioning 5
physical files *(continued)*
  data type 23
  DBCS considerations 84
  decimal positions 27
  field name 8
  form type 5
  key field name 8
  keyword entries 30
  length 21
  location 30
  name 6
  name type 5
  positional entries 4
  record format name 6
  reference 20
  reserved 6
  sequence number 5
  specification type 5
  Unicode 82
  usage 28
positional entries for physical and logical files 4

R

RANGE (Range) keyword 65
record format 7
record format name 6
REF (Reference) keyword 67
REFACCPTH (Reference Access Path Definition) keyword 67
reference 20
Reference Access Path Definition (REFACCPTH) keyword 67
Reference Shift (REFSHIFT) keyword 69
Referenced Field (REFFLD) keyword 68
REFFLD (Referenced Field) keyword 68
REFSHIFT (Reference Shift) keyword 69
RENAME (Rename) keyword 70
reserved 6

S

select or omit-field level
  specifying COMP 40
select/omit field name 18
sequence number 5
SIGNED (Signed) keyword 71
simple logical files 2
specification type 5
specifying COMP at the select or omit-field level 40
SST (Substring) keyword 72
Substring (SST) keyword 72

T

TEXT (Text) keyword 74
Time Format (TIMFMT) keyword 74
Time Separator (TIMSEP) keyword 75
TIMFMT (Time Format) keyword 74
TIMSEP (Time Separator) keyword 75
Translation Table (TRNTBL) keyword 76
TRNTBL (Translation Table) keyword 76

U

Unicode
    DDS file considerations 82
    physical and logical files 82
UNIQUE (Unique) keyword 77
UNSIGNED (Unsigned) keyword 78
usage 28

V

VALUES (Values) keyword 79
Variable-Length Field (VARLEN) keyword 80
VARLEN (Variable-Length Field) keyword 80

Z

ZONE (Zone) keyword 81



Product Number: 5770-SS1