TPF Database Facility

Programming Concepts and Reference

Release 1

SH31-0179-08

TPF Database Facility

Programming Concepts and Reference

Release 1

SH31-0179-08

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page xi.

Ninth Edition (October 2001)

This is a major revision of, and obsoletes, SH31-0179-07.

This edition applies to Version 1 Release 1 Modification Level 3 of IBM Transaction Processing Facility Database Facility, program number 5706-196, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

IBM welcomes your comments. Address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1997, 2001. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Tables ix

Notices xi
Programming Interface Information xi
Trademarks xii

About This Book xiii
Before You Begin xiii
Who Should Read This Book xiii
How This Book Is Organized xiii
Conventions Used in the TPFDF Library xiii
How to Read the Syntax Diagrams xiv
Related Information xvii
IBM TPF Database Facility (TPFDF) Books xvii
IBM Transaction Processing Facility (TPF) 4.1 Books xvii
IBM Airline Control System (ALCS) Books xviii
o Online Information xviii
How to Send Your Comments xviii

Part 1. Application Programming 1

Application Programming Overview 3
Files and Subfiles 3
Data Level Usage 3
Using Extended Logical Records 4
Specifying Algorithm Arguments with TPFDF Macros and Functions 4
Using Basic Indexing with Macros and Functions 5
Using Keys in a Detail File 5
Adding LRECs to Detail Files Using Basic Indexing 5
Grouping LRECs Together Using the Unique Key Facility 6
Using Unique Keys 6
Transaction Manager 7
ALCS Support 8
Commit Scopes 8
Programming Conventions 8
Internal Use of Commit Scopes 10
Checkpoint and Close Processing 10
Benefits of Using Commit Scopes 10

Identifying Return Indicators and Errors 13
Checking for Errors in Assembler Using Equates 14
Checking for Errors Using Structured Programming Macros (SPMs) 14
Checking for Errors Using C Functions 14
Format 15
Normal Return 16
Examples 17

Specifying Logical Records (LRECs) Using Keys 19
Overview of Keys 19
Differences between Key Lists and Key*n* Parameters 19
Using Keys When Reading LRECs 20

© Copyright IBM Corp. 1997, 2001

Reading LRECs Using Partial Keys 20
Reading LRECs Using a B+Tree Index 21
Using Variable-Length Fields as Keys 22
Specifying File Organization with Key*n* Parameters 23
Key*n* Parameters Examples 24
Setting Up and Using a Key List 26
Setting Up a Key List 26
Using a Key List 27
Using Boolean Logic in Key Lists 30
Using Default-Key Key Lists 31
Using Modification Key Lists 31

Sample Applications 33
Problem and Solution 33
Member File Definitions DSECT 35
Assembler Application Program Example 39
Processing the Member File Using TPFDF Macros. 39
C Language Application Program Example. 48
Processing the Member File Using TPFDF C Functions 51

Part 2. C Language Functions 69

TPFDF General-Use C Language Functions: Reference 71
dfadd–Add a Logical Record to a Subfile 73
dfadr–Provide the File Address of a Prime Block 80
dfckp–Checkpoint a Subfile 83
dfclr–Allow ECB Exit with Open Files. 85
dfcls–Close a Subfile. 86
dfcpy–Copy a Subfile 92
dfcre–Create a Subfile 95
dfdel–Delete One or More Logical Records 97
dfdix–Delete Index References to a Subfile 105
dfdsp–Display Logical Records from a Subfile 107
dffrl–Ensure an ECB Data Level Is Free 111
dfidx–Create an Index Reference. 112
dfifb–Check a SW00SR Slot 114
dfkey–Activate a Key List. 115
dfmod–Perform or Indicate Logical Record Modifications 117
dfmrg–Merge Logical Records from Two Subfiles 121
df_nbrkeys–Setting Up the Number of Keys 124
dfopn–Open a Subfile 125
dfopt–Set Optional Information. 130
dfred–Read a Logical Record 134
dfrep–Replace a Logical Record with Another Logical Record 143
dfret–Retain a Logical Record Position. 145
dfrst–Restore a Subfile 147
df_setkey–Setting Up a Key in a Key List. 150
dfspa–Create Work Space 156
dfsrt–Sort a Subfile 157
dftld–Write a Subfile from Main Storage to DASD. 160
dftlg–Write a File or Subfile to Tape 163
dftrd–Read a Subfile from an Input Tape to Main Storage. 166
dfuky–Generate a Unique Key for Use in Logical Records 167
member_size–Calculating the Size of a Structure Member 168

TPFDF Restricted C Language Functions: Reference 169

dftab–Access Database Definition Tables 170

Part 3. Assembler Macros 173

TPFDF General-Use Assembler Macros: Reference 175
DBADD–Add a Logical Record to a Subfile 176
DBADR–Provide the File Address of a Prime Block 190
DBCKP–Checkpoint a Subfile 195
DBCLR–Allow ECB Exit with Open Files 198
DBCLS–Close a Subfile 199
DBCPY–Copy a Subfile 205
DBCRE–Create a Subfile 210
DBDEL–Delete One or More Logical Records 214
DBDIX–Delete Index References to a Subfile 228
DBDSP–Display Logical Records from a Subfile 231
DBFRL–Ensure an ECB Data Level Is Free 240
DBIDX–Create an Index Reference 241
DBIFB–Check a SW00SR Slot 244
DBKEY–Activate a Key List 247
DBMOD–Perform or Indicate Logical Record Modifications 249
DBMRG–Merge Logical Records from Two Subfiles 254
DBOPN–Open a Subfile 260
DBRED–Read a Logical Record 272
DBREP–Replace a Logical Record with Another Logical Record 286
DBRET–Retain a Logical Record Position 290
DBRST–Restore a Subfile 293
DBSETK–Setting Up a Key in a Key List 298
DBSPA–Create Work Space 304
DBSRT–Sort a Subfile 307
DBTLD–Write a Subfile from Main Storage to DASD 313
DBTLG–Write a File or Subfile to Tape 318
DBTRD–Read a Subfile from an Input Tape to Main Storage 323
DBUKY–Generate a Unique Key for Use in Logical Records 325

TPFDF Restricted Assembler Macros: Reference 327
BLKSZ–Convert a Block Type to a Block Size 328
DBCNT–Calculate the Length of an Assembler Symbol 332
DBTAB–Access Database Definition (DBDEF) Tables 333
DFCAS–TPFDF Case Setup in Fast-link Segments 336
DFCLIB–C Language Interface 338
DFDDA–Distributed Data Access Support 341
DFDLAY–Delay Processing Conditionally 342
DFGDS–General Data Set Support User Exit 343
DFGETC–Get Working Storage Block 344
DFGLVL–Get Resource Level 346
DFGPNL–Get Calling Program Address 347
DFLNK–TPFDF Fast Linkage 348
DFSSU–Handling DBDEF Subtables 350
DFTDC–Dialogue Control Facility Support User Exit 353
DFUEX–Define TPFDF User Exit Point 354
FILTP–Determine File Address Type 355
FMSGS–Set Up Output Messages 357
HELPA–Help Message Text 360

Index 363

Figures

1. How SubLRECs Are Added to an Extended LREC 4
2. How subLRECs Are Numbered in an Extended LREC 4
3. Example of LRECs Related by Unique Key Fields 7
4. LRECs Organized on Four Key Fields 20
5. LREC Data Organized on Four Key Fields 21
6. LRECs on a Keyed File 22
7. Key Fields in LRECs without a Terminating Character 23
8. Key Fields in LRECs with a Terminating Character 23
9. IR00DF–Member File Definitions DSECT 35
10. SAM0–File Maintenance Program 40
11. SAM–Departure Control Interface 45
12. SAM2–Monthly Maintenance Program 47
13. ir00df.h–Passenger Record Structure Declaration 49
14. psgr.h–Member File Definitions Header 51
15. sam0.c–File Maintenance Program 52
16. sam1.c–Departure Control Interface 63
17. sam2.c–Monthly Maintenance Program 65
18. Merging LRECs from Two Subfiles 122
19. Sorting LRECs from One Subfile into Another 159
20. Merging LRECs from Two Subfiles 258
21. Sorting LRECs from One Subfile into Another 312
22. Areas in the Database Definition (DBDEF) Table Accessed by the DBTAB Macro. 334
23. DFSSU Macro: Create a Copy of the TPFDF and COMMON Subtables 350

© Copyright IBM Corp. 1997, 2001

Tables

1. Begin and End Transactions 7
2. Suspend and Resume Transactions 7
3. Error Conditions 13
4. Serious Errors 13
5. Equates For TPFDF Macro Error Checking 14
6. C Functions for Detecting Errors 14
7. Values for the cond Parameter 16
8. Normal Return Values for Error Detection C Functions 16
9. Boolean Equates 30

© Copyright IBM Corp. 1997, 2001

Notices

References in this book to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service in this book is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give 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
USA

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
Department 830A
Mail Drop P131
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Programming Interface Information

This book is intended to help you write application programs that run under the IBM TPF Database Facility (TPFDF) product. This book primarily documents General-use Programming Interface and Associated Guidance Information provided by the TPFDF product.

General-use programming interfaces allow the customer to write programs that obtain the services of the TPFDF product.

However, this book also documents Product-sensitive Programming Interface and Associated Guidance Information provided by the TPFDF product.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of the TPFDF product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be

© Copyright IBM Corp. 1997, 2001

expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Product-sensitive programming interface

Product-sensitive Programming Interface and Associated Guidance Information...

End of Product-sensitive programming interface

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM
S/390

Other company, product, and service names may be trademarks or service marks of others.

About This Book

This book contains programming concepts and a reference for the C language functions and assembler macros that you can use when writing application programs for the IBM Transaction Processing Facility Database Facility (TPFDF).

In this book, abbreviations are often used instead of spelled-out terms. Every term is spelled out at first mention followed by the all-caps abbreviation enclosed in parentheses; for example, structured programming macro (SPM). Abbreviations are defined again at various intervals throughout the book. In addition, the majority of abbreviations and their definitions are listed in the master glossary in *Messages (System Error, Online, Offline) and Master Glossary*.

Before You Begin

Before using this book, see *TPFDF General Information* for an overall understanding of the TPFDF product. In addition, you must be familiar with the way your database is designed. See *TPFDF Database Administration* and your database administrator for information about defining the files and subfiles for your environment.

Who Should Read This Book

This book is intended for application programmers who are currently working with one of the following:

- Transaction Processing Facility (TPF) system and IBM High Level Assembler/MVS & VM & VSE (HLASM)
- Airline Control System (ALCS), also referred to as TPF/MVS, and IBM Assembler H or IBM High Level Assembler/MVS & VM & VSE (HLASM).

How This Book Is Organized

This book has three parts as follows:

- Part 1, “Application Programming” provides some programming concepts for writing TPFDF application programs as well as a sample application program in assembler and in C language.
- Part 2, “C Language Functions” provides detailed information about the general-use and restricted C functions.
- Part 3, “Assembler Macros” provides detailed information about the general-use and restricted assembler macros.

Conventions Used in the TPFDF Library

The TPFDF library uses the following conventions:

Typography	Examples of Usage
italic	Used for important words and phrases. For example: A *database* is a collection of data. Used to represent variable information. For example: Enter **ZUDFC DISPLAY ID-***fileid*, where *fileid* is the file identifier (ID) of the file for which you want statistics.

© Copyright IBM Corp. 1997, 2001

Typography	Examples of Usage
bold	Used to represent keywords. For example: Enter **ZUDFC HELP** to obtain help information for the ZUDFC command.

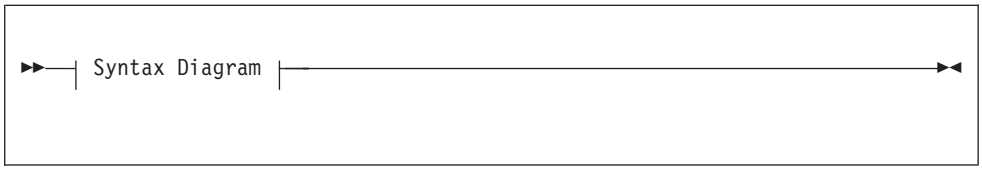
`monospaced`	Used for messages and information that displays on a screen. For example:
	`PROCESSING COMPLETED`
	Used for C language functions. For example:
	`dfcls`
	Used for examples. For example:
	`ZUDFC DISPLAY ID-J5`

bold italic	Used for emphasis. For example: You ***must*** type this command exactly as shown.
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=*option*

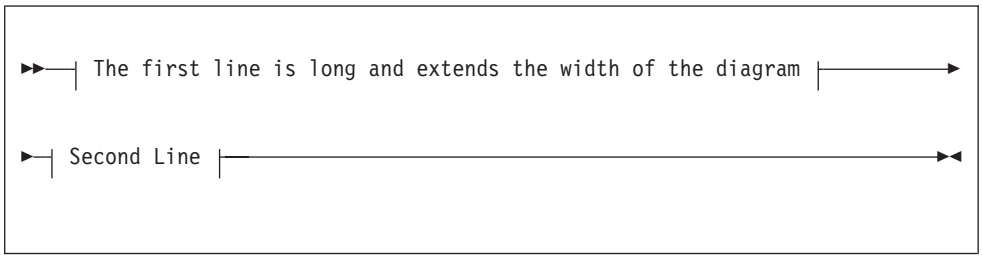
How to Read the Syntax Diagrams

This section describes how to read the syntax diagrams (informally called *railroad tracks*) used in this book.

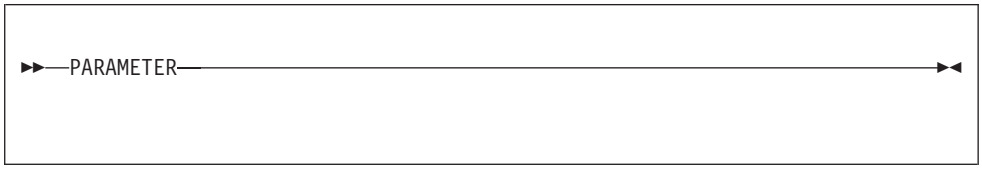
- Read the diagrams from left-to-right, top-to-bottom, following the main path line. Each diagram begins on the left with double arrowheads and ends on the right with 2 arrowheads facing each other.



- If a diagram is longer than one line, the first line ends with a single arrowhead and the second line begins with a single arrowhead.

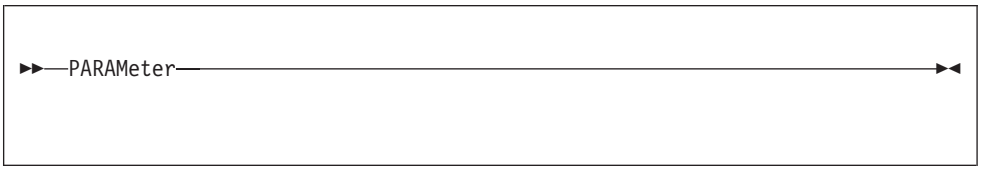


- A word in all uppercase is a parameter that you must spell ***exactly*** as shown.

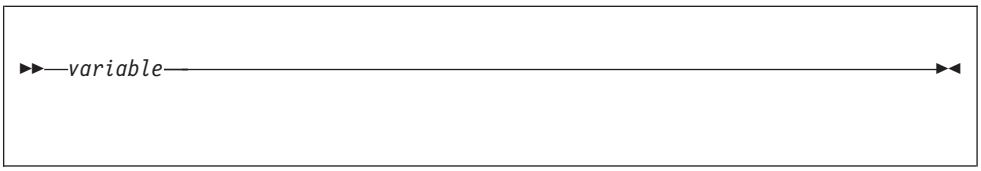


- If you can abbreviate a parameter, the optional part of the parameter is shown in lowercase. (You must type the text that is shown in uppercase. You can type none, one, or more of the letters that are shown in lowercase.)

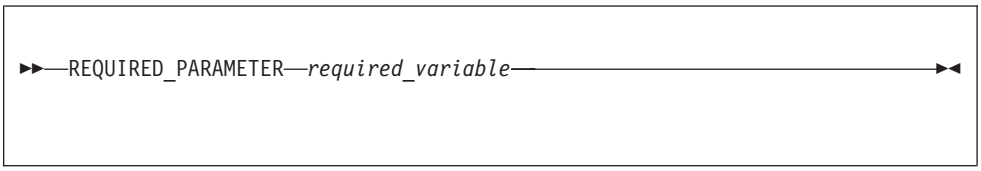
Note: Some TPF commands are case-sensitive and contain parameters that must be entered exactly as shown. This information is noted in the description of the appropriate commands.



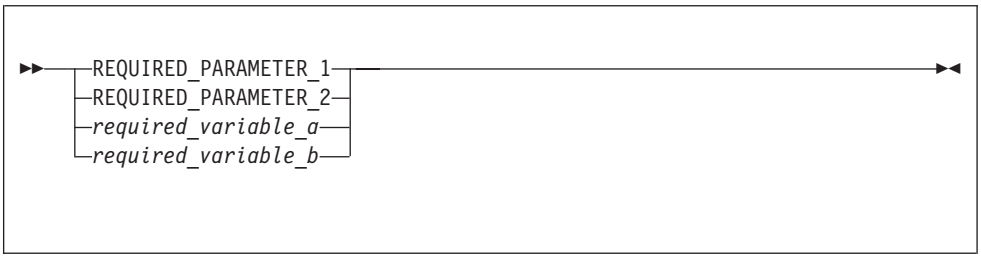
- A word in all lowercase italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.



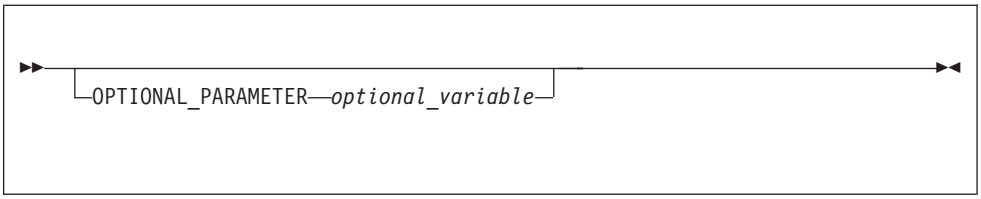
- Required parameters and variables are shown on the main path line. You must code required parameters and variables.



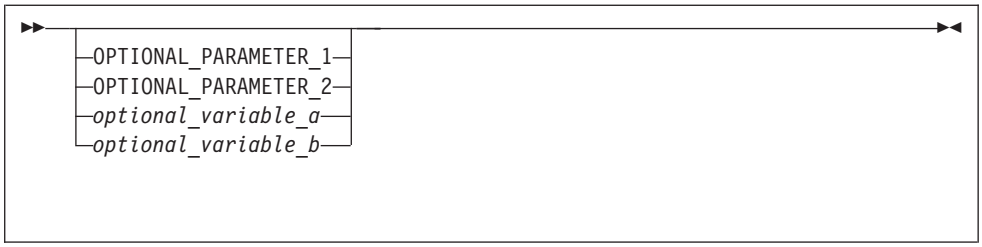
- If there is more than one mutually exclusive required parameter or variable to choose from, they are stacked vertically.



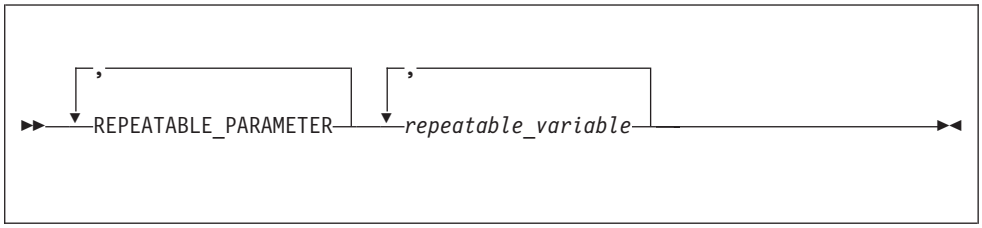
- Optional parameters and variables are shown below the main path line. You can choose not to code optional parameters and variables.



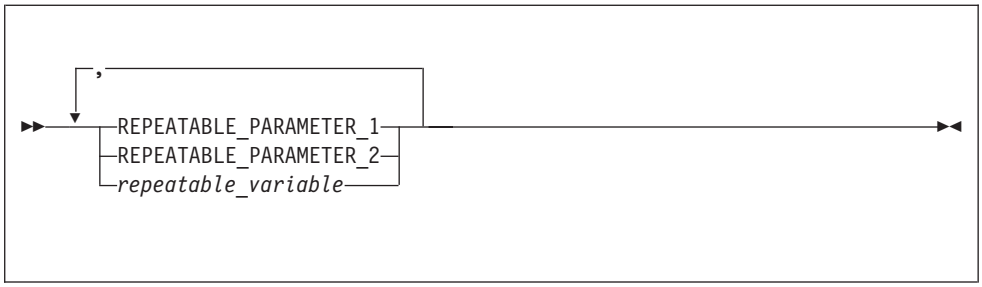
- If there is more than one mutually exclusive optional parameter or variable to choose from, they are stacked vertically below the main path line.



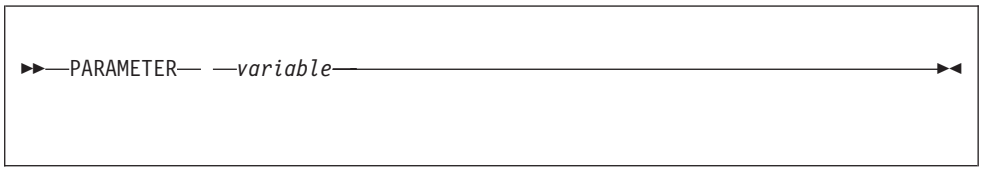
- An arrow returning to the left above a parameter or variable on the main path line means that the parameter or variable can be repeated. The comma (,) means that each parameter or variable must be separated from the next parameter or variable by a comma.



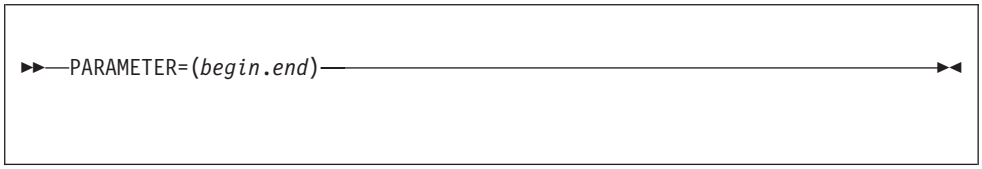
- An arrow returning to the left above a group of parameters or variables means that more than one can be selected, or a single one can be repeated.



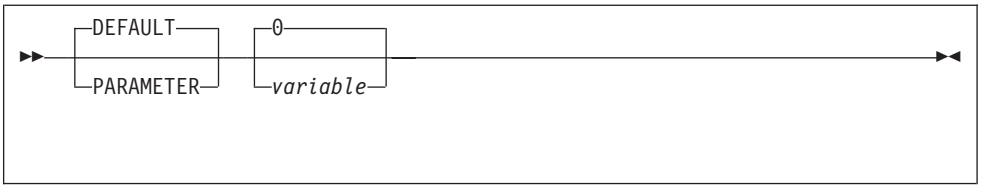
- If a diagram shows a blank space, you must code the blank space as part of the syntax. In the following example, you must code **PARAMETER** *variable*.



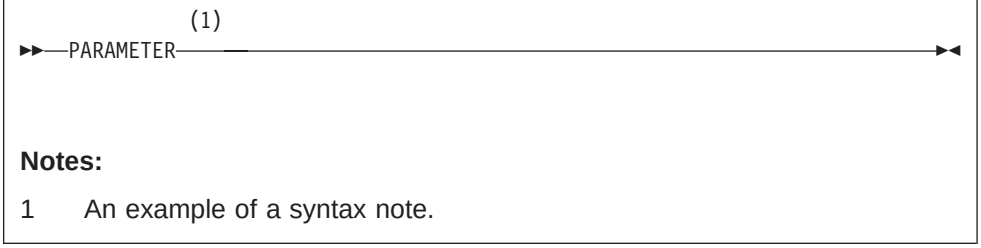
- If a diagram shows a character that is not alphanumeric (such as commas, parentheses, periods, and equal signs), you must code the character as part of the syntax. In the following example, you must code **PARAMETER=(***begin***.***end***)**.



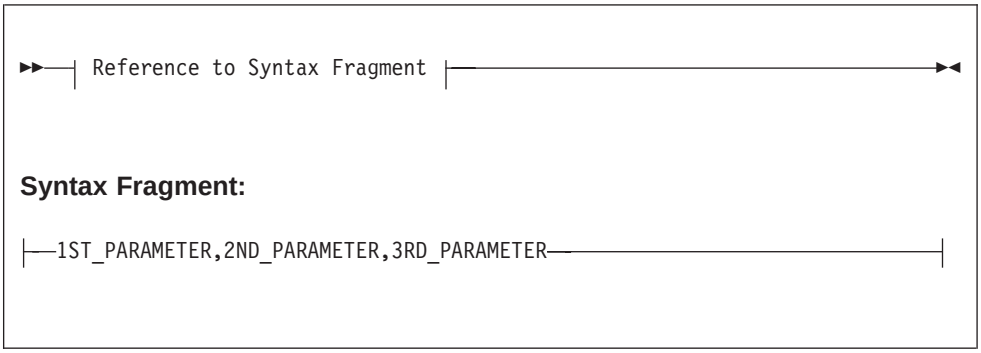
- Default parameters and values are shown above the main path line. The TPF system uses the default if you omit the parameter or value entirely.



- References to syntax notes are shown as numbers enclosed in parentheses above the line. Do not code the parentheses or the number.



- Some diagrams contain *syntax fragments*, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram and in the heading of the fragment. The fragment is placed below the main diagram.



Related Information

A list of related information follows. For information on how to order or access any of this information, call your IBM representative.

IBM TPF Database Facility (TPFDF) Books

- *TPFDF Database Administration*, SH31-0175
- *TPFDF General Information*, GH31-0177
- *TPFDF Installation and Customization*, GH31-0178
- *TPFDF and TPF Structured Programming Macros*, SH31-0183.

IBM Transaction Processing Facility (TPF) 4.1 Books

- *TPF Application Programming*, SH31-0132
- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF Database Reference*, SH31-0143.

IBM Airline Control System (ALCS) Books

- *ALCS Application Programming Guide*, SH19-6948.

Online Information

- *Messages (System Error, Online, Offline) and Master Glossary*.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TPF information, use one of the methods that follow. Make sure you include the title and number of the book, the version of your product and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send your comments electronically, do either of the following:
  - Go to http://www.ibm.com/tpf/pubs/tpfpubs.htm.

    There you will find a link to a feedback page where you can enter and submit comments.
  - Send your comments by e-mail to tpfid@us.ibm.com
- If you prefer to send your comments by mail, address your comments to:

  IBM Corporation
  TPF Systems Information Development
  Mail Station P923
  2455 South Road
  Poughkeepsie, NY 12601-5400
  USA
- If you prefer to send your comments by FAX, use this number:
  - United States and Canada: 1 + 845 + 432 + 9788
  - Other countries: (international code) + 845 + 432 +9788

Part 1. Application Programming

Application Programming Overview . . . 3
Files and Subfiles . . . 3
Data Level Usage . . . 3
Using Extended Logical Records . . . 4
Specifying Algorithm Arguments with TPFDF Macros and Functions . . . 4
Using Basic Indexing with Macros and Functions . . . 5
  Using Keys in a Detail File . . . 5
  Adding LRECs to Detail Files Using Basic Indexing . . . 5
Grouping LRECs Together Using the Unique Key Facility . . . 6
  Using Unique Keys . . . 6
Transaction Manager . . . 7
  ALCS Support . . . 8
  Commit Scopes . . . 8
    Root Commit Scopes . . . 8
    Nested Commit Scopes . . . 8
    Suspended Commit Scopes . . . 8
  Programming Conventions . . . 8
  Internal Use of Commit Scopes . . . 10
  Checkpoint and Close Processing . . . 10
  Benefits of Using Commit Scopes . . . 10

Identifying Return Indicators and Errors . . . 13
Checking for Errors in Assembler Using Equates . . . 14
Checking for Errors Using Structured Programming Macros (SPMs) . . . 14
Checking for Errors Using C Functions . . . 14
  Format . . . 15
  Normal Return . . . 16
  Examples . . . 17

Specifying Logical Records (LRECs) Using Keys . . . 19
Overview of Keys . . . 19
  Differences between Key Lists and Key*n* Parameters . . . 19
Using Keys When Reading LRECs . . . 20
  Reading LRECs Using Partial Keys . . . 20
  Reading LRECs Using a B^+Tree Index . . . 21
Using Variable-Length Fields as Keys . . . 22
Specifying File Organization with Key*n* Parameters . . . 23
  Key*n* Parameters Examples . . . 24
Setting Up and Using a Key List . . . 26
  Setting Up a Key List . . . 26
  Using a Key List . . . 27
    Processing Using Key*n* Parameters . . . 28
    Processing Using a Key List . . . 28
  Using Boolean Logic in Key Lists . . . 30
  Using Default-Key Key Lists . . . 31
  Using Modification Key Lists . . . 31

Sample Applications . . . 33
Problem and Solution . . . 33
Member File Definitions DSECT . . . 35
Assembler Application Program Example . . . 39
  Processing the Member File Using TPFDF Macros . . . 39
    File Maintenance Program . . . 40

© Copyright IBM Corp. 1997, 2001

Departure Control Interface Program 45
Monthly Maintenance Program 47
C Language Application Program Example. 48
Processing the Member File Using TPFDF C Functions 51
File Maintenance Program. 52
Departure Control Interface Program 63
Monthly Maintenance Program 65

Application Programming Overview

The TPF Database Facility (TPFDF) provides an interface between application programs requesting access to information in a database and the system software that physically accesses the stored data. The main interfaces between application programs and the TPFDF product are provided through a set of C functions and a set of assembler macros. See Part 2, “C Language Functions” for details about the TPFDF C functions and Part 3, “Assembler Macros” for details about the TPFDF macros. In addition, see “Sample Applications” on page 33 for sample application programs using the TPFDF C functions and TPFDF macros.

See *TPFDF General Information* for an overall understanding of the TPFDF product.

In addition, work with your database administrator to ensure you know how the files in your database are defined. See *TPFDF Database Administration* for more information about how files are defined for a TPFDF database.

This chapter provides an overview of some of the general considerations for writing application programs for the TPFDF product.

Files and Subfiles

Most of the macros and functions operate on logical records (LRECs) in a subfile. Some macros and functions operate on one or more subfiles, and some on whole files.

When you call a C function, you identify the subfile you want to access by providing a pointer to the SW00SR slot created when you call the `dfopn` function. When you call a macro, you identify the file you want to access by its DSECT name and you specify the subfile by using the ALG, FILE, or ORD parameter.

Data Level Usage

The TPFDF product preserves all data levels across TPFDF macro calls with the following exceptions:

- If you use the DBOPN macro with the DATA and PARAM parameters specified, the data is returned to the data level specified.

  Note: The DATA and PARAM parameters are provided for migration purposes only.
- If you use the DBDSP macro:
  - Data level 1 (D1) and data level 3 (D3) are ***not*** data level independent (DLI) if the WTOPC parameter is specified with the NO value (the default), or the YES value is not specified, and DBLCL macro symbol &ACPDBAA is set to zero.
  - Data level 2 (D2) is ***not*** DLI.
- If you use the `dfdsp` function:
  - Data level 1 (D1) and data level 3 (D3) are ***not*** DLI if you do not specify the DFDSP_WTOPC value and DBLCL macro symbol &ACPDBAA is set to zero.
  - Data level 2 (D2) is ***not*** DLI.

© Copyright IBM Corp. 1997, 2001

Using Extended Logical Records

In application programs, you can do the following with extended logical records (LRECs):

- Add an extended LREC, including the userLREC, to a subfile
- Add a subLREC to an extended LREC
- Read a complete extended LREC, including the userLREC and all subLRECs
- Delete one or more subLRECs from an extended LREC
- Replace the subLREC or userLREC (or both) in an extended LREC.

The TPFDF product adds each subLREC immediately after the control part of the extended LREC. If you add several subLRECs to the same extended LREC, the subLRECs appear in reverse order in the LREC. This is shown in Figure 1.

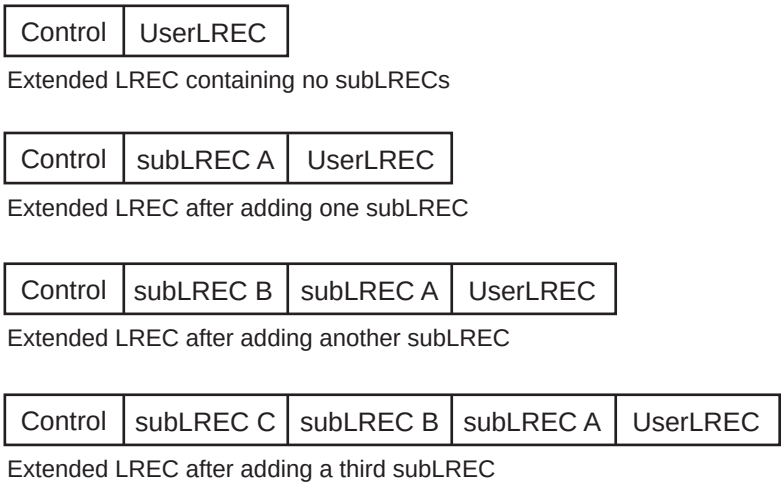


Figure 1. How SubLRECs Are Added to an Extended LREC

The subLRECs are numbered in the order they appear in the extended LREC (see Figure 2).

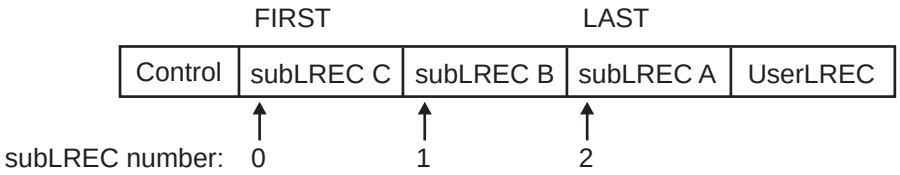


Figure 2. How subLRECs Are Numbered in an Extended LREC

See *TPFDF Database Administration* for more information about defining extended LRECs.

Specifying Algorithm Arguments with TPFDF Macros and Functions

When you call a TPFDF macro or C function, you can use an algorithm argument to identify the particular subfile in the file that you are accessing. The way you specify an algorithm argument depends on the type of algorithm that you have defined in the DSECT macro for the file. See *TPFDF Database Administration* for more information about algorithms, and see “TPFDF General-Use C Language Functions:

Reference” on page 71 and “TPFDF General-Use Assembler Macros: Reference” on page 175 for more information about specifying algorithm arguments with TPFDF functions and macros.

Using Basic Indexing with Macros and Functions

Applications normally perform all actions on the *detail file*, not on the index file, when they use basic indexing.

For example, to read an LREC from a detail file using the TPFDF macros and basic indexing:

- Open the detail file normally using the REF parameter. For example:

  ```
  DBOPN REF=GR25DF
  ```

- Read an LREC using DBRED. Provide the index key of an index LREC as an algorithm argument in the ALG parameter. For example, to read the LREC in the detail file for a passenger with the name of ADLER, you can code:

  ```
  DBRED REF=GR25DF,ALG==C'ADLER'
  ```

See *TPFDF Database Administration* for more information about basic indexing and the other forms of TPFDF index support.

Using Keys in a Detail File

An index LREC only points to the prime block of a subfile. The ALG parameter only defines the detail subfile that contains the LREC you require.

To locate a particular LREC in the subfile, you must also supply keys with the KEY*n* parameters or a key list. For example, to read an LREC with an LREC ID defined by equate #GR25K80 for passenger JONES, you can code:

```
DBRED REF=GR25DF,ALG==C'JONES',KEY1=(PKY=#GR25K80)
```

See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about keys.

Adding LRECs to Detail Files Using Basic Indexing

Using basic indexing, you can add new LRECs to a detail file using the DBADD macro or `dfadd` function. Specify the *index key* as the algorithm argument. The TPFDF product inserts the LREC in the appropriate place in the detail file.

For example, to add a passenger named Jones to the passenger file, you can code:

```
DBADD REF=GR23DF,ALG==C'JONES',NEWLREC=EBW060
```

In this example, EBW060 contains the LREC to be added with information about passenger Jones.

If the detail subfile is organized UP or DOWN, specify KEY*n* parameters or a key list to place the LREC in an appropriate position in the detail subfile.

If the detail subfile does not exist, the TPFDF product will create the subfile and the index references automatically. For example, to create a detail subfile, the index references to that subfile, and add a passenger named Jones, you can code:

```
DBADD REF=GR23DF,INDEX,ALG==C'JONES',NEWLREC=EBW060
```

This has the same effect as the following sequence of macros:

```
DBCRE REF=GR23DF
DBIDX REF=GR23DF,ALG==C'JONES'
DBADD REF=GR23DF,NEWLREC=EBW060
```

Grouping LRECs Together Using the Unique Key Facility

Suppose you want to store a large quantity of data about customers by using different LREC types for different parts of the data.

For example, suppose the information is to be stored in a file using three different types of LRECs:

LREC ID	Information held
X'80'	Customer name, held in a variable-length field called *zzzz*NAM.
X'90'	Customer address.
X'A0'	Additional information, only applicable for certain customers.

You need some method of indicating that all the LRECs for a particular customer are related. You can implement this requirement in a variety of ways. A simple method is to repeat the name field (as a fixed-length field) in each LREC, but this can waste a lot of space. A better way is to use the unique keys.

Using Unique Keys

You can use unique keys to indicate that LRECs are related. To do this, define a unique key field in each LREC and call the DBUKY macro or `dfuky` function to provide you with a unique key to insert in this field in each LREC.

The process is as follows:

1. Before adding a new customer name LREC to the file, call the DBUKY macro or `dfuky` function to get a new unique key. The TPFDF product places this 4-byte key in a field in SW00SR slot SW00UKY.
2. Store the unique key in a user field of each related LREC (for example, all the LRECs relating to a particular customer).
3. Ensure the DSECT has the additional 18-byte header extension.

The result is shown in Figure 3 on page 7. One 4-byte unique key field exists in each LREC. This example shows the unique keys in the same relative position in each LREC, but this is not necessary.

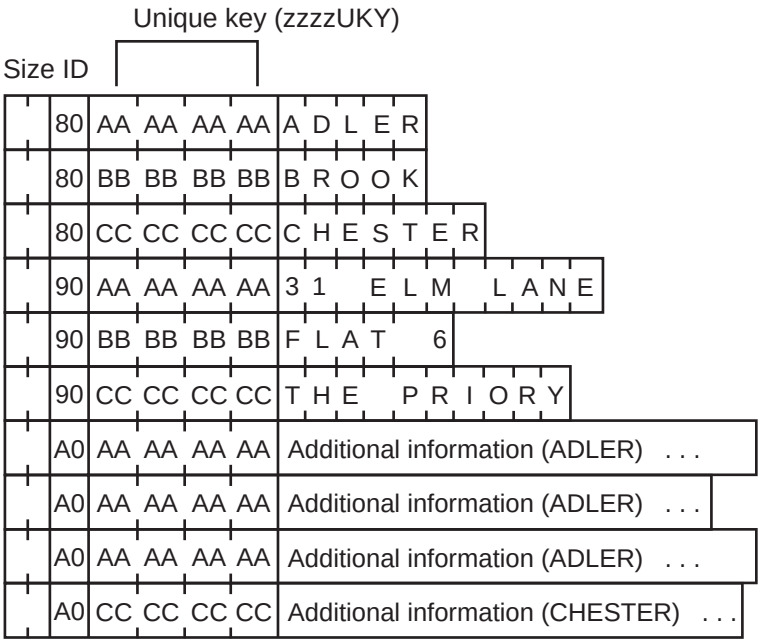


Figure 3. Example of LRECs Related by Unique Key Fields

Transaction Manager

TPFDF product support of the TPF transaction manager (TM) ensures a consistent view of a database in an application program. This consistent view of the database ensures that either all or none of the file changes have been completed; there is no such thing as a partially updated database.

The TM verifies a consistent database view by using a set of application programming interfaces (APIs) for the application program to define both the scope of a file transaction as well as actions to be taken for the file transaction.

The TM provides a subset of the TX functions (defined by the X/Open TX interface) to the application to begin and end (that is, commit or roll back) a transaction:

Table 1. Begin and End Transactions

C Function	Assembler Macro
`tx_begin`	TXBGC
`tx_commit`	TXCMC
`tx_rollback`	TXRBC

Additionally, the TM provides the following extension (to X/Open) functions to the application to suspend or resume a transaction:

Table 2. Suspend and Resume Transactions

C Function	Assembler Macro
`tx_suspend_tpf`	TXSPC
`tx_resume_tpf`	TXRSC

ALCS Support

The ALCS environment does not support the TM.

Commit Scopes

The TM groups all database updates into a unit of work called a *commit scope*. A *current* commit scope is defined by a file transaction.

A file transaction refers to all macros, instructions, and functions that are issued on a file. When a file transaction is completed, you can save (commit) or discard (roll back) the file updates.

Root Commit Scopes

The first commit scope that is created by the application is a *root commit scope*.

Nested Commit Scopes

A commit scope that is created after the root scope has been activated is a *nested commit scope*. A root commit scope can have many nested commit scopes.

In a commit scope, all database requests are satisfied back through the chain of nested scopes to the root commit scope. Therefore, none of the nested commit scopes are satisfied until the root commit scope is satisfied.

Suspended Commit Scopes

You can temporarily suspend a commit scope to make database updates outside the current commit scope. However, when a commit scope is suspended, you cannot reference a file that is already referenced in the current commit scope.

See the following for more information about the transaction manager or commit scopes:

- *TPF Application Programming*
- *TPF Database Reference*
- *TPF C/C++ Language Support User's Guide*.

Programming Conventions

The TM APIs can be used with TPFDF macros in an application program to ensure that all file updates are complete before being committed (saved) to the database. All updates are filed to DASD as one update; that is, all or none of the updates take effect. TPFDF macros that update the database do not take effect on DASD until a commit is issued for the root commit scope. If a rollback is issued for the root commit scope, all updates made by TPFDF macro calls in the commit scope are discarded.

The following programming conventions provide you with safe and simple use of commits scopes with TPFDF macros:

- All references to a TPFDF file, from open to close, must be made in the same commit scope. If a TPFDF file was opened in a commit scope (root or at any nested level), all references to the file (including the closing) must be done in that commit scope.
- The rollback of a commit scope discards all updates made as a result of TPFDF macro calls in that commit scope. You can issue a rollback at any point in the macro sequence. It is not necessary that a file be closed after a rollback because it discards the open and any subsequent macros.

Attention: A rollback has no effect on any updates (nonfile updates) issued outside of the TPFDF macros; for example, application use of the data levels, registers, and local variables.

- You cannot reference a TPFDF file that is opened in a suspended commit scope.
- TPFDF macros that reference core resident files (specifically W-type files in detac mode and T-type files) are not affected by commit scopes. This means that updates to these files cannot be rolled back.
- Avoid TPFDF transactions that access and modify large amounts of data because of the impacts to TM resources.
- TPFDF detac mode supports files of all sizes and is available in an ALCS environment. With TPFDF detac mode, you can roll back a file update without having to access the commit scope or roll back the commit scope.

The following shows an example of a basic TPFDF transaction that is contained in a commit scope. The TXBGC macro begins the commit scope that contains the DBOPN, DBADD, and DBCLS macros issued on FILE1. The TXCMC macro ends the commit scope and commits FILE1 to the database.

```
TXBGC

DBOPN REF=FILE1

DBADD REF=FILE1

DBCLS REF=FILE1

TXCMC
```

The following shows an example of TPFDF transactions contained in a root commit scope and a nested commit scope. The first TXBGC macro begins the root commit scope that contains the DBOPN and DBRED macros issued on FILE1. The second TXBGC macro begins the nested commit scope that contains the DBOPN, DBADD, and DBCLS macros issued on FILE2 and FILE3. Although the nested commit scope ends with the subsequent TXCMC macro, it is not until the root commit scope is ended with the TXCMC macro that FILE1, FILE2, and FILE3 are committed to the database because nested commit scopes are not satisfied until the root commit scope is satisfied. Additionally, a file that is outside a nested commit scope cannot be referenced from within a nested commit scope. If FILE1 is accessed within the nested commit scope, an error would occur.

```
TXBGC

DBOPN REF=FILE1

DBRED REF=FILE1

TXBGC

DBOPN REF=FILE2

DBOPN REF=FILE3

DBADD REF=FILE2

DBADD REF=FILE3

DBCLS REF=FILE2

DBCLS REF=FILE3

TXCMC
```

```
DBCLS REF=FILE1

TXCMC
```

Internal Use of Commit Scopes

To provide added security against database corruption that results from unplanned system outages, the TM support of TPFDF transactions includes an option to use commit scopes internally. This option is valuable when many files are to be filed out during checkpoint and close processing (for example, detac mode, extensive B^+Tree indexing updates, and requests to pack indexes).

Checkpoint and Close Processing

You can control the use of commit scopes internally by specifying the optional parameters for the TPFDF macros that use checkpoint and close processing. These optional parameters can override the database definition (DBDEF) default setting to enable or disable the internal use of commit scopes. Using commit scopes provides protection against database corruption, such as when system outages occur during the large filing of updates.

See the following for additional information about the option to manage TPFDF macros with commit scopes:

- “DBCKP–Checkpoint a Subfile” on page 195
- “DBCLS–Close a Subfile” on page 199
- “dfcls–Close a Subfile” on page 86
- “dfckp–Checkpoint a Subfile” on page 83.

See *TPFDF Database Administration* for additional information about the DBDEF option to manage TPFDF macros with commit scopes.

Benefits of Using Commit Scopes

Using commit scopes with TPFDF macros provides the following benefits:

- ***Database consistency and integrity***

  Using commit scopes ensures either all or none of the files updated in a database are written to DASD. There is no such thing as a partially updated database when updates are all made in commit scopes.
- ***Managing TPFDF DASD-resident files and traditional database transactions***

  Commit scopes support both TPFDF and traditional files, including critical and internal files; that is, a commit scope can contain updates to both TPFDF and traditional files.
- ***Managing application programs***

  Using commit scopes simplifies your application program designs with a reduced need for error recovery routines, and shortens your coding and testing phases. The reliability of your application program is increased because the order of updates is no longer of paramount importance. At any time, the application program could stop processing and request that the TPF system ignore all of its previous updates. Additionally, knowing that you always have a consistent view of the database affects your application development cycle time. Either all of your file changes have been completed or none of them have; it is that simple.
- ***Increasing application program reliability***

The order of updates is no longer of paramount importance. At any time, the application program could stop processing and request that the TPF system ignore all of its previous updates.

Identifying Return Indicators and Errors

After you call a macro or function, the TPFDF product indicates the success or failure of the call by setting indicator bits in the SW00RTN field in the SW00SR slot for that subfile. The SW00RT1 and SW00RT2 fields provide additional information about error conditions.

When a macro or function ends successfully, the TPFDF product sets SW00RTN to zero and the application program continues normally.

Table 3 shows the SW00RTN and SW00RT2 settings that indicate the application program must take some remedial action.

Table 3. Error Conditions

SW00SR Error Bits	Condition
SW00RTN #BIT1	One of the following: • The logical record (LREC) was not found when using a DBRED macro, `dfred` function, DBDEL macro, or `dfdel` function. • An LREC was found with the same keys as the LREC being added by one of the following: – DBADD macro with the UNIQUE parameter specified – `dfadd` function with the DFADD_UNIQUE value specified for the *options* parameter – DBADD macro or `dfadd` function for a file with UNIQUE=YES specified on the DBDEF macro for the file.
SW00RTN #BIT5	End of file occurred during fullfile processing.
SW00RT2 #BIT0	Error in list of logical record numbers. This bit is only meaningful if bit 1 in SW00RTN is ON.
SW00RT2 #BIT5	The subfile is empty because of the following: • The next available byte (NAB) value in the prime block indicates the first byte directly after the header. • There are no forward chains in the prime block.

Table 4 shows the SW00RTN and SW00RT2 settings that indicate more serious errors. The TPFDF product may issue a system error when one of these errors occur.

Table 4. Serious Errors

SW00SR Error Bits	Condition
SW00RTN #BIT0	I/O or B^{+}Tree index error.
SW00RTN #BIT2	File address compute program (FACE) error.
SW00RTN #BIT3	One of the following: • Subfile does not exist in detail file or intermediate index file. • Incorrect algorithm argument.
SW00RTN #BIT4	Data in block is corrupt.
SW00RTN #BIT6	Sequence error using a DBRST macro or `dfrst` function.
SW00RTN #BIT7	Sort or merge error.
SW00RT2 #BIT1	Error with the DBDSP macro or `dfdsp` function.

© Copyright IBM Corp. 1997, 2001

Table 4. Serious Errors (continued)

SW00SR Error Bits	Condition
SW00RT2 #BIT4	B+Tree index error.

When you use fullfile processing, SW00RT1 contains the number of errors detected since the file was opened.

Checking for Errors in Assembler Using Equates

Table 5 lists a set of equates for some of the common errors that you can experience. You can use these equates to test the value of SW00RTN.

Table 5. Equates For TPFDF Macro Error Checking

Equate	Hexadecimal Value	Condition
#TPFDBEX	X'AB'	Any serious error indicated in SW00RTN except indexing errors.
#TPFDBER	X'BB'	Any serious error indicated in SW00RTN.
#TPFDBNR	X'44'	Any error indicated in SW00RTN requiring remedial action.
#TPFDBNI	X'44'	Any error indicated in SW00RTN requiring remedial action.
#TPFDBOK	X'00'	No errors indicated in SW00RTN.

Checking for Errors Using Structured Programming Macros (SPMs)

You can also use a *short form* method to check return codes by using a set of special parameters with the SPMs.

To check SW00RTN return codes, use the following parameters with the SPMs:
- DBFOUND
- DBERROR
- DBEOF
- DBIDX.

To check a specific SW00RT2 return code, use the DBEMPTY parameter with the SPMs.

See *TPFDF and TPF Structured Programming Macros* for more information about these parameters and how to use them.

Checking for Errors Using C Functions

Table 6 lists a set of C functions that you can use to test a particular condition when you have called a C function.

Note: Do not use any of these functions after a `dfcls` function.

Table 6. C Functions for Detecting Errors

Function	Condition
`DF_OK`	No errors indicated in SW00RTN.

Table 6. C Functions for Detecting Errors (continued)

Function	Condition
DF_EMPTY	Test if a subfile is empty. This function is available only after a delete operation that does not use fullfile processing and before the next TPFDF call.
DF_ER	Any serious error indicated in SW00RTN.
DF_ERBTR	B+Tree index error.
DF_ERX	Any serious error indicated in SW00RTN except indexing errors.
DF_NR	Record not found (any error indicated in SW00RTN requiring remedial action).
DF_EF	End of file (any error indicated in SW00RTN requiring remedial action).
DF_ERCNT	Numbers of errors detected during fullfile processing.
DF_ERLST	Error in list of logical record numbers.
DF_ERDSP	Error with the dfdsp function.
DF_SERRC	System error issued by the TPFDF product.
DF_TEST	Test specified bits in SW00RTN.

The following describes the format, parameters, and normal return for these functions. In addition, there is an example of how you can use these functions.

Format

```
int DF_OK(dft_fil *file);
int DF_EMPTY(dft_fil *file);
int DF_ER(dft_fil *file);
int DF_ERBTR(dft_fil *file);
int DF_ERX(dft_fil *file);
int DF_NR(dft_fil *file);
int DF_EF(dft_fil *file);
int DF_ERCNT(dft_fil *file);
int DF_ERLST(dft_fil *file);
int DF_ERDSP(dft_fil *file);
int DF_SERRC(dft_fil *file);
int DF_TEST(dft_fil *file, char cond);
```

cond Parameter Values:

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

cond
is the condition that you want to test. Table 7 describes the values you can specify:

Table 7. Values for the cond Parameter

Value	SW00RTN Bit Tested	Condition
DFC_IOERR	Bit 0	I/O or B+Tree index error.
DFC_RCDNF	Bit 1	One of the following: • The LREC was not found when using a `dfred` or `dfdel` function. • An LREC was found with the same keys as the LREC being added by one of the following: – `dfadd` function with the DFADD_UNIQUE value specified for the *options* parameter – `dfadd` function for a file with UNIQUE=YES specified on the DBDEF macro for the file.
DFC_FACE	Bit 2	File address compute program (FACE) error.
DFC_ALG	Bit 3	One of the following: • Subfile does not exist in detail file or intermediate index file. • Incorrect algorithm argument.
DFC_BLOCK	Bit 4	Data in block is corrupt.
DFC_EOF	Bit 5	End of file occurred during fullfile processing.
DFC_SEQ	Bit 6	Sequence error using `dfrst` function.
DFC_SM	Bit 7	Sort or merge error.

Normal Return

Function `DF_ERCNT` returns the count of errors detected during fullfile processing. All other functions return a nonzero value if the tested condition is true; otherwise, the return is zero. Table 8 on page 17 lists the conditions tested for each function.

Table 8. Normal Return Values for Error Detection C Functions

Function	Tested Condition
DF_OK	All SW00RTN bits are zero.
DF_ER	One of the following SW00RTN bits is set to 1: • #BIT0 • #BIT2 • #BIT3 • #BIT4 • #BIT6 • #BIT7.
DF_ERBTR	Both SW00RTN #BIT0 and SW00RT2 #BIT4 are set to 1.
DF_ERX	One of the following SW00RTN bits is set to 1: • #BIT0 • #BIT2 • #BIT4 • #BIT6 • #BIT7.
DF_NR	One of the following SW00RTN bits is set to 1: • #BIT1 • #BIT5.
DF_EF	One of the following SW00RTN bits is set to 1: • #BIT1 • #BIT5.
DF_ERLST	Both SW00RTN #BIT1 and SW00RT2 #BIT0 are set to 1.
DF_ERDSP	Both SW00RTN #BIT0 and SW00RT2 #BIT1 are set to 1.
DF_SERRC	One of the following SW00RTN bits is set to 1: • #BIT0 • #BIT2 • #BIT4 • #BIT7.

Examples

The following example:

- Reads an LREC from a passenger detail subfile referenced by the member number contained in a variable (character array) called in_mess.
- Tests for a read error.
- Tests if a particular subfile exists. If the subfile does not exist, a message is sent to the operator.

```
psgr_lrec = dfred_acc(psgr_file, DFRED_ALG, 0, in_mess);

if (DF_ERX(psgr_file))                /* if error from read       */
  subfile_error();                    /* perform error processing */

if (DF_TEST(psgr_file, DFC_ALG))      /* if subfile doesn't exist */
  printf("INVALID NUMBER\n");         /* send message to operator */
```

Specifying Logical Records (LRECs) Using Keys

Once you identify the subfile you are going to work with, you may want to identify an LREC or group of LRECs in the subfile. You can do this by using keys or by specifying LREC sequence numbers, or both. This chapter discusses various aspects of using key parameters; see the individual macro and function descriptions for details about the specific key parameters and LREC sequence number parameters.

Overview of Keys

Specifying keys includes information such as the displacement and length of the key fields, a comparison operator, and the values to compare the fields against. By using keys, you can identify an LREC or group of LRECs in a subfile that match certain criteria. Keys become active when you:

- Specify KEY*n* parameters on a macro.
- Use a key list with the DBKEY macro or `dfkey` function, or with the KEYLIST parameter on a macro.
- Use default keys with the DBADD macro or `dfadd` function.

Active keys remain in effect and are used by the TPFDF product while the file is open until one of the following conditions occurs.

- If you specify different keys using one of the previously mentioned methods, the new keys will be used and remain in active.
- If you use the NOKEY, FAST, or INLINE parameter with another macro that accesses the subfile, any currently active keys are deactivated.
- If you use the DF*xxx*_NOKEY, DF*xxx*_FAST, or DF*xxx*_INLINE value of the *options* parameter for another function that accesses the subfile, any currently active keys are deactivated (*xxx* represents the function; for example, DFADD_NOKEY).
- If you code a DBADD macro or `dfadd` function and the database definition (DBDEF) has default keys specified, the default keys become the active keys.
- If you specify a DBREP macro or `dfrep` function and KEYCHECK=YES is defined on the DBDEF macro, the active keys cannot be predicted.

Differences between Key Lists and Key*n* Parameters

Key lists and the KEY*n* parameters provide the same function, but the following operations are only available with key lists:

- Default keys on read operations, or any macro or function that implies a read operation (for example, the DBDEL macro and `dfdel` function read the record before deleting it).
- Boolean OR logic with keys.
- Global modification of LRECs.

In addition, the following are other differences between key lists and KEY*n* parameters:

- You can specify as many as 180 keys with a key list, but you can specify only as many as 6 keys with the KEY*n* parameters.
- Key lists generate the active keys at run time, but the KEY*n* parameters generate the active keys at assembly time.

© Copyright IBM Corp. 1997, 2001

See the individual macro descriptions for details about the format of the KEY*n* parameters. See “Setting Up and Using a Key List” on page 26 for more information about using key lists.

Using Keys When Reading LRECs

When keys are active on a read operation, the TPFDF product searches all the LRECs in the subfile (depending on the options specified) and returns the first LREC with key fields that match the ones specified in the active keys. The DBRED macro and `dfred` function are not the only macro and function that perform read operations. Other macros and functions perform read operations during internal processing. For example, the DBDEL macro and `dfdel` function both do a read operation to locate the correct LREC and then they do the delete operation.

If the same set of keys is used on multiple read operations, they do not have to be specified on each macro or function. Once activated by a macro or function, keys remain active and can be used by subsequent macros or functions (except as noted in “Overview of Keys” on page 19). Instead of specifying keys each time you read or delete LRECs, you can specify them when you open a subfile or use the DBKEY macro or `dfkey` function. If you then read LRECs without specifying any more key parameters, the TPFDF product supplies only those LRECs that match the keys specified with the DBOPN macro, `dfopn` function, DBKEY macro, or `dfkey` function.

Reading LRECs Using Partial Keys

You can read LRECs using just some of the keys that are used to organize the LRECs in each subfile. This is known as using a *partial key*.

For example, assume that the LRECs shown in Figure 4 are UP organized on fields PKY and FLD1, DOWN organized on field FLD2, and UP organized on field FLD3.

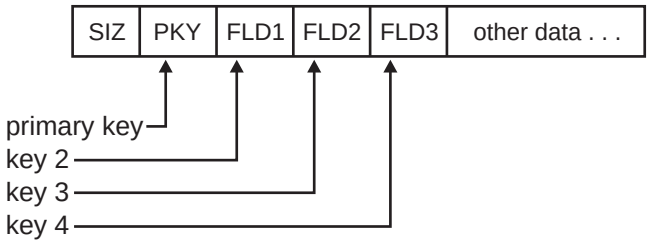


Figure 4. LRECs Organized on Four Key Fields

Figure 5 on page 21 shows the sample contents of such a file.

```
PKY     FLD1      FLD2      FLD3
80      1000      4000      2000
80      1000      4000      4000
80      1000      3000      1000
80      1000      3000      2000
80      2000      3000      3000
80      2000      3000      4000
80      2000      2000      2000
80      3000      3000      1000
80      3000      3000      3000
80      3000      3000      5000
80      3000      1000      1000
80      3000      1000      4000
80      5000      4000      2000
80      5000      4000      4000
```

Figure 5. LREC Data Organized on Four Key Fields

The TPFDF product lets you read LRECs without specifying all the keys used in the file organization. You can specify only a selection of the key fields in the search. For example, you could use fields PKY, FLD1 and FLD3. However, because you are omitting one of the key fields in the search argument, the LRECs must not be treated as UP or DOWN organized on any of the less-significant key fields (in this case, field FLD3).

It is essential that you define the organization of the LRECs on these less-significant key fields as NOORG. Otherwise, you will not locate all the LRECs matching your criteria. For example:

```
DBRED REF=zzzzzz,                                  *
      KEY1=(PKY=#zzzzK80,UP),                      *
      KEY2=(R=zzzzFLD1,S=EBW000,UP),               *
      KEY3=(R=zzzzFLD3,S=EBX000,NOORG)
```

Specifying an organization of NOORG (or not specifying organization) on a file that has organization can cause the TPFDF product to unnecessarily search for more LRECs that match the selection criteria. However, the same LRECs will be retrieved whether organization is specified or not.

Reading LRECs Using a B+Tree Index

Large subfiles can require many DASD accesses for the TPFDF product to search for a particular LREC. DASD accesses can be minimized by using B+Tree indexing.

The TPFDF product uses a B+Tree index to quickly find an LREC in a B+Tree data file. The TPFDF product uses the B+Tree index while reading a B+Tree data file when the active keys match the order, organization, and displacement of the default keys. The active keys are used until one does not match the default keys.

In the following example, KEY1, KEY2, and KEY4 in the DBRED statement match KEY1, KEY2, and KEY4 in the DBDEF statement, but KEY3 does not match. In this example, the TPFDF product uses the B+Tree index to locate the first LREC matching KEY1 and KEY2. The TPFDF product then searches sequentially through the subfile until it finds an LREC matching KEY3 and KEY4.

```
DBDEF FILE=GR91SR,
      KEY1=(PKY=#GR91K80,UP),
      KEY2=(R=GR91FLD1,UP),
      KEY3=(R=GR91FLD3,DOWN),
      KEY4=(R=GR91FLD4,UP)

DBRED REF=GR91SR,
```

```
KEY1=(PKY=#GR91K80,UP),
KEY2=(R=GR91FLD1,S=EBW000,UP),
KEY3=(R=GR91FLD6,S=EBX000,DOWN),
KEY4=(R=GR91FLD4,S=EBW044,UP)
```

Note: The TPFDF product will not use a key to search the B+Tree index if the key was defined with one of the following:

- Using the M or D subparameter with a KEY*n* parameter
- Using a key list when SW01ID1 #BIT5 or #BIT6 is set to 1.

The TPFDF product will use only a previously active key to search the B+Tree index. This key and any subsequent keys are then used by the TPFDF product to sequentially search for the requested LREC.

You can read LRECs specifying only the part of the key that is used to organize the file. For example, suppose we have a file that is organized in the ascending order of seat number in a flight number as shown in Figure 6.

```
    LREC ID  Flight      Seat      Customer
    (Hex.)   number      number    Name

      80     1233        A99       xxxxxx
S     80     1234        C31       xxxxxxxxxx
S     80     1234        C45       xxxxxxxxxxxx
S     80     1234        D17       xxxxxxx
S     80     1234        E05       xxxxxxxxxx
      80     1236        A02       xxxxxx
      80     1236        D01       xxxxxx
      ----------------------
      key of the file (ascending)
```

Figure 6. LRECs on a Keyed File

To extract the customers travelling on flight 1234, you could request the TPFDF product to read LRECs with a primary key of X'80' and a secondary key (flight number) of 1234. The TPFDF product would give you the first LREC marked with an S. On subsequent reads it gives you the next LREC and so on until it has given you all the matching LRECs.

Now assume that the file uses B+Tree indexing and the flight and seat number were coded as default keys in the DBDEF, but only flight number 1234 is supplied when you read the file. The first time DBRED is issued, the B+Tree index is used to find the first LREC with flight number 1234. The next time DBRED is issued, DBRED does not use the B+Tree index to find the next LREC with flight number 1234. The file is searched sequentially.

Applications do not have to be updated to use B+Tree indexing. B+Tree indexing provides a more effective method for accessing LRECs in large files that have unusual, unknown, or changing distributions.

Using Variable-Length Fields as Keys

You can use variable-length fields as key fields by indicating how much of the field must match. You can do this by specifying an L subparameter with a KEY*n* parameter or with field SW01LEN in a key list. However, using variable-length fields as keys can result in false matches. For example, if you use the string "ADAMS" as

a search key with a length of 5, the TPFDF product matches all the LRECs shown in Figure 7.

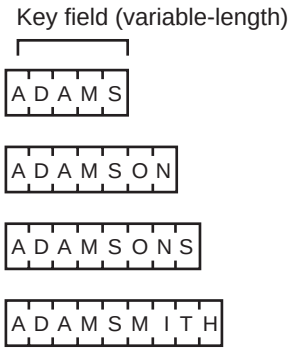


Figure 7. Key Fields in LRECs without a Terminating Character

One method of preventing false matches is to use a terminating character at the end of the variable-length field when each LREC is added to the file. If X'00' was used as the terminating character in our example, the entries shown in Figure 7 now become those shown in Figure 8.

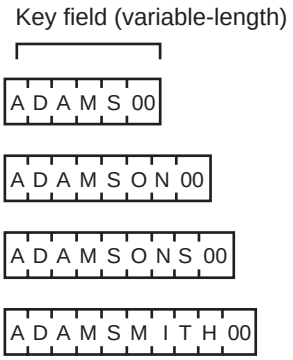


Figure 8. Key Fields in LRECs with a Terminating Character

To find a match for ADAMS, you must add a X'00' character to the end of the search field.

Specifying File Organization with Key*n* Parameters

If you specify the KEY*n* parameters on a macro, you must also specify the organization, if there is any, in one of the following ways:

- Specify a general organization that applies to all keys using the UP, DOWN, or NOORG parameter on the macro. If you use this method, no individual key on that macro can specify its own organization.

  If you specify an M or D subparameter with a KEY*n* parameter on the macro, that key and all subsequent keys will have an organization of NOORG.
- Specify the organization for each individual key using the organization parameters available with the KEY*n* parameter. If you specify NOORG for a particular key, all subsequent keys must specify NOORG.

  In addition, you must specify an organization of NOORG for any key that uses the M or D subparameter with the KEY*n* parameter.

If you do not want to assign any organization to the keys, it is possible to have no organization parameters at all.

If the organization is not specified according to these rules, the TPFDF product issues one or more assembly messages (called MNOTEs). If an MNOTE is issued, the code is generated based on the following rules:

1. If an organization is specified, that organization is used.
2. Otherwise, if an organization for KEY1 is specified, the KEY1 organization is used.
3. Otherwise, if a general organization is specified, that organization is used.
4. Otherwise, NOORG is used for the organization.

Exceptions:

1. If a previous key organization was specified as NOORG, this key becomes NOORG as well.
2. If the key uses the M or D subparameter, NOORG is used for the organization and an organization of NOORG is assumed for any subsequent key.

Migration Note

Any applications that do not currently follow the rules for specifying the organization for keys do not have to be changed until they are reassembled. In addition, you can control the severity of the MNOTEs that are issued when the rules are not followed. We strongly recommend that you use the default severity of 8; however, if you are confident that your applications are working as designed without following the organization rules, you can choose to ignore the following MNOTEs (where *n* represents the key number):

```
KEYn ORG NOT ALLOWED IF GENERAL ORG GIVEN
KEYn REQUIRES ORG IF NO GENERAL ORG GIVEN
KEYn HAS ORG, BUT NO KEY1 ORG OR GENERAL ORG
UP/DOWN ON KEYn NOT ALLOWED AFTER PREVIOUS NOORG - NOORG ASSUMED
```

The severity is controlled with the &KMNVAL variable in the DBLCL macro. See *TPFDF Installation and Customization* for more information about the DBLCL macro and associated variables.

In addition, the TPFDF product issues an informational MNOTE at the end of the macro expansion that summarizes the defined key organization.

Key*n* Parameters Examples

The following are examples of macros that are coded by following these key organization rules. The examples also show the resulting MNOTEs.

Correct Example 1

```
 DBRED REF=GR95SR,REG=R5,DOWN,
       KEY1=(PKY=#GR95K80),
       KEY2=(R=#GR95SEX,S=EBX004),
       KEY3=(R=#GR95AGE,S=EBX008)

Result: *,KEY1 - KEY3 DEFINED: K1-DW K2-DW K3-DW
```

End of Correct Example 1

Correct Example 2

```
 DBRED REF=GR95SR,REG=R5,
       KEY1=(PKY=#GR95K80,UP),
       KEY2=(R=#GR95SEX,S=EBX004,DOWN),
       KEY3=(R=#GR95AGE,S=EBX008,NOORG)

Result: *,KEY1 - KEY3 DEFINED: K1-UP K2-DW K3-NO
```

End of Correct Example 2

Correct Example 3

```
 DBRED REF=GR95SR,REG=R5,
       KEY1=(PKY=#GR95K80,DOWN),
       KEY2=(R=#GR95SEX,M=X'80',C=Z,NOORG),
       KEY3=(R=#GR95AGE,S=EBX008,NOORG)

Result: *,KEY2 TM: NOORG ASSUMED FOR THIS AND FOLLOWING KEYS
        *,KEY1 - KEY3 DEFINED: K1-DW K2-NO K3-NO
```

End of Correct Example 3

The following are examples of macros that are coded without following the organization rules. The examples also show the resulting MNOTEs.

Incorrect Example 1

```
 DBRED REF=GR95SR,REG=R5,DOWN,
       KEY1=(PKY=#GR95K80,UP),
       KEY2=(R=#GR95SEX,S=EBX004),
       KEY3=(R=#GR95AGE,S=EBX008,NOORG)

Result: 8,KEY1 ORG NOT ALLOWED IF GENERAL ORG GIVEN
        8,KEY3 ORG NOT ALLOWED IF GENERAL ORG GIVEN
        *,KEY1 - KEY3 DEFINED: K1-UP K2-UP K3-NO
```

End of Incorrect Example 1

Incorrect Example 2

```
 DBRED REF=GR95SR,REG=R5,
       KEY1=(PKY=#GR95K80,UP),
       KEY2=(R=#GR95SEX,S=EBX004),
       KEY3=(R=#GR95AGE,S=EBX008,DOWN)

Result: 8,KEY2 REQUIRES ORG IF NO GENERAL ORG GIVEN
        *,KEY1 - KEY3 DEFINED: K1-UP K2-UP K3-DW
```

End of Incorrect Example 2

Incorrect Example 3

```
 DBRED REF=GR95SR,REG=R5,
       KEY1=(PKY=#GR95K80,UP),
       KEY2=(R=#GR95SEX,M=X'80'),
       KEY3=(R=#GR95AGE,S=EBX008,DOWN)

Result: 8,KEY2 REQUIRES ORG IF NO GENERAL ORG GIVEN
        *,KEY2 TM: NOORG ASSUMED FOR THIS AND FOLLOWING KEYS
        8,UP/DOWN ON KEY3 NOT ALLOWED AFTER PREVIOUS NOORG
        *,KEY1 - KEY3 DEFINED: K1-UP K2-NO K3-NO
```

End of Incorrect Example 3

Setting Up and Using a Key List

A *key list* contains information that allows the TPFDF product to search for logical records (LRECs) by comparing criteria supplied by the application program with specified data fields in the LREC. The criteria specified in a key list is the same criteria that you can specify with the KEY*n* parameters; that is:

- The displacement and length of the LREC key fields
- A comparison operator
- The values to compare the fields against.

As mentioned previously, there are certain operations that you can only use with key lists, as follows:

- Using default keys on read operations
- Using Boolean operators with keys
- Global modification of LRECs
- Using more than six keys to search a TPFDF database.

The following are the different types of key lists:

- Selection key list
- Default-key key list
- Modification key list
- Sort/merge key list.

A *selection key list* is used to specify criteria used when searching a subfile. Only LRECs that match the criteria will be processed.

A *default-key key list* is used to specify which set of keys defined in the DBDEF should be used when searching a subfile. A prototype LREC with target values for the key fields is used to locate LRECs that match the criteria, and only those LRECs are processed.

A *modification key list* is used to define rules for updating LRECs in a subfile. Fields in LRECs in a subfile are updated based on values and modification operations specified in the key list.

A *sort/merge key list* is used to specify how to sort LRECs into the output file on a DBSRT or DBMRG macro, or a `dfsrt` or `dfmrg` function.

Setting Up a Key List

You can set up a key list in one of the following ways:

- Specify the number of keys in field SW01NKY of the SW01SR DSECT by using an assembler instruction and then use the DBSETK macro to set up each key.

  See “Using a Key List with the DBSETK Macro” on page 28 for an example of how to use this method.
- Specify the number of keys by using the `df_nbrkeys` function and then use the `df_setkey` function to set up each key.

  See “Using a Key List with the `df_setkey` Function” on page 29 for an example of how to use this method.
- Use assembler instructions to fill in the key list structure, which is defined by the SW01SR DSECT.

  See “Processing Using a Key List” on page 28 for an example of how to do this.

Note: This method is provided only for compatibility with older applications; do not use this method for new applications or when updating existing applications.

The first two bytes of a key list (field SW01NKY) specifies the number of keys in the key list. The maximum value for SW01NKY differs for each type of key list as follows:

Type of Key List	Maximum Number of Keys
Selection key list	180 keys
Default-key key list	1 key
Modification key list	6 keys
Sort/merge key list	180 keys

Field SW01NKY is followed by the keys, each of which is 12 bytes, as follows:

Field	Number of Bytes	Description
SW01DIS	2 bytes	Displacement into the LREC
SW01LEN	2 bytes	Length of the key
SW01CON	1 byte	Condition that must exist for the match to be successful
SW01MSK	1 byte	One of the following: • 1-byte mask • Default key LREC ID, used with default-key key lists.
SW01SEA	4 bytes	One of the following: • Search argument • Pointer to prototype LREC, used with default-key key lists.
SW01ID1	1 byte	Option indicators
SW01ID2	1 byte	One of the following: • Boolean connector • Modification operation, used with modification key lists.

See “DBSETK–Setting Up a Key in a Key List” on page 298 and “df_setkey–Setting Up a Key in a Key List” on page 150 for information about the values that you can specify for fields SW01CON, SW01ID1, and SW01ID2.

Using a Key List

The following example describes one possible use of a key list for the DBRED macro. Use the same coding techniques, as appropriate, for the other macros or functions that allow the use of key lists.

In many instances an application has to verify the database according to given conditions. The more options that are available, the more DBRED macros using KEY*n* parameters are necessary to handle the requests. Consequently, the program becomes unstructured and generates many bytes of object code for each DBRED macro.

The key list technique allows you to define active keys at run time according to specific needs. The following example shows the advantages of this technique.

Processing Using Key*n* Parameters

If, for example, you want to compare field EBW000(4) with field IR73FLD at displacement 6 in an LREC without using a key list, you have to call a DBRED macro for each condition compared:

```
#IF CLC MI0ACC+10(2),EQ,=C'GT'
   DBRED REF=IR73DF,KEY1=(PKY=#IR73K80),                        *
         KEY2=(R=IR73FLD,S=EBW000,C=GT),UP
#ELIF CLC,M10ACC+10(2),EQ,=C'GE'
   DBRED REF=IR73DF,KEY1=(PKY=#IR73K80),                        *
         KEY2=(R=IR73FLD,S=EBW000,C=GE),UP
#ELIF CLC,M10ACC+10(2),EQ,=C'EQ'
   DBRED REF=IR73DF,KEY1=(PKY=#IR73K80),                        *
         KEY2=(R=IR73FLD,S=EBW000,C=EQ),UP
#ELIF CLC,M10ACC+10(2),EQ,=C'NE'
   DBRED REF=IR73DF,KEY1=(PKY=#IR73K80),                        *
         KEY2=(R=IR73FLD,S=EBW000,C=NE),UP
#ELIF CLC,M10ACC+10(2),EQ,=C'LT'
   DBRED REF=IR73DF,KEY1=(PKY=#IR73K80),                        *
         KEY2=(R=IR73FLD,S=EBW000,C=LT),UP
#ELIF CLC,M10ACC+10(2),EQ,=C'LE'
   DBRED REF=IR73DF,KEY1=(PKY=#IR73K80),                        *
         KEY2=(R=IR73FLD,S=EBW000,C=LE),UP
#EIF
```

Processing Using a Key List

If you set up a key list, the amount of object code generated is greatly reduced. As mentioned previously, you can set up the key list by manually coding the key list structure or by using the DBSETK macro or `df_setkey` function.

The following shows the DBRED example using the manual method:

```
SW01SR   REG=R5
LA   R5,EBX000                    LOAD BASE OF KEY LIST
XC   EBX000(L'SW01NKY+2*L'SW01KIT),EBX000  CLEAR KEY LIST AREA
MVC  SW01NKY,=H'2'                SET UP 2 KEYS
MVC  SW01DIS,=H'2'                DISPLACEMENT TO PRIMARY KEY
MVC  SW01LEN,=H'1'                PKY LENGTH=1
MVI  SW01CON,#DF_EQ               SET EQ MATCH
MVI  SW01MSK,#IR73K80             SET MASK=PKY
MVI  SW01ID1,#DF_UP+#DF_CONST     UP ORG + USE CLI WITH MASK
MVI  SW01ID2,#DF_AND              AND CONNECTOR
LA   R5,L'SW01KIT(,R5)            SET TO SECOND KEY
MVC  SW01DIS,=H'6'                DISPLACEMENT TO SECOND KEY
MVC  SW01LEN,=H'4'                LENGTH OF KEY
LA   R14,EBW000                   ADDR OF 2ND KEY SEARCH ARG
ST   R14,SW01SEA                  AND STORE
MVI  SW01ID1,#DF_UP               UP ORGANIZATION
#IF    MI0ACC+10(2),EQ,=C'GT'     INPUT = GREATER THAN?
  MVI    SW01CON,#DF_GT           SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'GE'     INPUT = GREATER OR EQUAL?
  MVI    SW01CON,#DF_GE           SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'EQ'     INPUT = EQUAL?
  MVI    SW01CON,#DF_EQ           SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'NE'     INPUT = NOT EQUAL?
  MVI    SW01CON,#DF_NE           SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'LT'     INPUT = LESS THAN?
  MVI    SW01CON,#DF_LT           SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'LE'     INPUT = LESS OR EQUAL?
  MVI    SW01CON,#DF_LE           SET UP CONDITION CODE
#EIF
DBKEY  REF=IR73DF,KEYLIST=EBX000  ACTIVATE THE KEY LIST
DBRED  REF=IR73DF
```

Using a Key List with the DBSETK Macro: The following shows the DBRED example using the DBSETK macro:

```
SW01SR REG=R5
LA     R5,EBX000                    LOAD BASE OF KEY LIST
XC     EBX000(L'SW01NKY+2*L'SW01KIT),EBX000   CLEAR KEY LIST AREA
MVC    SW01NKY,=H'2'                SET UP 2 KEYS
DBSETK BASE=R5,KEYNUM=1,DIS=I/2,LEN=I/1,CON=#DF_EQ,MSK=#IR73K80,  *
      ID1=#DF_UP+#DF_CONST,ID2=#DF_AND
DBSETK BASE=R5,KEYNUM=2,DIS=I/6,LEN=I/4,SEA=EBW000,ID1=#DF_UP
LA     R5,L'SW01KIT(,R5)            POINT TO SECOND KEY SET
#IF    MI0ACC+10(2),EQ,=C'GT'       INPUT = GREATER THAN?
  DBSETK CON=#DF_GT                 SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'GE'       INPUT = GREATER OR EQUAL?
  DBSETK CON=#DF_GE                 SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'EQ'       INPUT = EQUAL?
  DBSETK CON=#DF_EQ                 SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'NE'       INPUT = NOT EQUAL?
  DBSETK CON=#DF_NE                 SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'LT'       INPUT = LESS THAN?
  DBSETK CON=#DF_LT                 SET UP CONDITION CODE
#ELIF  MI0ACC+10(2),EQ,=C'LE'       INPUT = LESS OR EQUAL?
  DBSETK CON=#DF_LE                 SET UP CONDITION CODE
#EIF
DBKEY  REF=IR73DF,KEYLIST=EBX000    ACTIVATE KEY LIST
DBRED  REF=IR73DF
```

Using a Key List with the `df_setkey` Function: The following example uses the `df_nbrkeys` and `df_setkey` functions to create a key list. The first key is used for the LREC ID, and the second key searches field ir73fld for string 1234567890.

```
/* Set up the key list with 2 key                              */
df_nbrkeys(&keylist,2);

/* Define the two keys using the df_setkey function            */
df_setkey(&keylist,1,offsetof(struct ir73df,ir73key),1,DF_EQ,
         0,_IR73K80,DF_UPORG,DF_CONST);

df_setkey(&keylist,2,offsetof(struct ir73df,ir73fld),10,DF_EQ,
         "1234567890",0,DF_UPORG,0);

/* Activate the key list, and read the record from the subfile */
dfkey(ir73_ptr,&keylist);
dfred(ir73_ptr,DFRED_BEGIN);
```

Setting up a Key List with Less than Six Keys: The following example shows how to set up a key list with less than six keys. A key list containing one key (the primary key of the LREC) is set up and then reads an LREC.

```
/* set up one key only (primary key of LREC) */

dft_pky pky = 0x80;
dft_kyl keys;
dft_rec *lrec91;

df_nbrkeys(&keys, 1);
df_setkey(&keys, 1, offsetoff(struct gr95sr, gr95key),
          1, DF_EQ, &pky, 0, DF_NOORG, DF_CHAR);

/* activate the key list */

dfkey(file_ptr, &keys);

/* read an LREC with matching primary key */
/* (start at the beginning of the subfile) */

lrec91 = dfred(file_ptr, DFRED_BEGIN);
```

Setting up a Key List in the Range 1-180: The following example shows how to set up a key list in the range 1-180. A key list containing seven keys is set up and then reads an LREC.

```
/* set up seven keys */

 dft_pky pky = 0x80;
 dft_kyl_ext keys;
 dft_rec *lrec91;

 df_setkey(&keys, 1, offsetoff(struct gr95sr, gr95key),
          1, DF_EQ, &pky, 0, DF_UPORG, DF_CHAR);
 df_setkey(&keys, 2, offsetoff(struct gr95sr, gr95f01),
          1, DF_EQ, &search_arg1, 0, DF_UPORG, DF_CHAR);
 df_setkey(&keys, 3, offsetoff(struct gr95sr, gr95f02),
          2, DF_EQ, &search_arg2, 0, DF_UPORG, DF_CHAR);
 df_setkey(&keys, 4, offsetoff(struct gr95sr, gr95f03),
          4, DF_EQ, &search_arg3, 0, DF_DOWNORG, DF_CHAR);
 df_setkey(&keys, 5, offsetoff(struct gr95sr, gr95f04),
          2, DF_EQ, &search_arg4, 0, DF_DOWNORG, DF_CHAR);
 df_setkey(&keys, 6, offsetoff(struct gr95sr, gr95f05),
          1, DF_NE, &search_arg5, 0, DF_NOORG, DF_CHAR);
 df_setkey(&keys, 7, offsetoff(struct gr95sr, gr95f06),
          1, DF_NE, &search_arg6, 0, DF_NOORG, DF_CHAR);

 /* activate the key list and set number of keys */

 dfkey_nbr(file_ptr, &keys, 7);

 /* read an LREC with matching keys */
 /* (start at the beginning of the subfile) */

 lrec91 = dfred(file_ptr, DFRED_BEGIN);
```

Using Boolean Logic in Key Lists

By default, each key specified in a key list or KEY*n* parameter is logically connected by AND Boolean logic. That is, each criteria must be satisfied for the LREC to be selected. Using key lists, additional Boolean logic can be specified that allows LRECs that meet some criteria but not necessarily other criteria to be selected. You can use any of the equates shown in Table 9 in SW01ID2 to control the connectivity between a key and the subsequent key:

Table 9. Boolean Equates

Assembler Connector	C Language Connector	Description
#DF_AND	DF_AND	Connects the current key to the next key using AND to form a group.
#DF_OR	DF_OR	Connects the current key to the next key using OR to form a group.
#DF_ANDIF	DF_ANDIF	Connects the current group to the next group using AND to form a complex expression.
#DF_ORIF	DF_ORIF	Connects the current group to the next group using OR to form a complex expression.

If any key specifies a Boolean connector value in field SW01ID2, all keys (except the last key in the key list) must specify Boolean values. Any Boolean connector value specified with the last key will be ignored. If none of these values are specified in field SW01ID2 of any key, all keys are connected using AND logic.

This support is not standard Boolean logic but is consistent with the implementation of the TPFDF structured programming macros (SPMs). See *TPFDF and TPF Structured Programming Macros* for more information.

Note: Using any of the Boolean connector values listed in Table 9 on page 30 in SW01ID2 has the following impact to LREC retrieval time (although the correct records will still be located):

- Although the organization of the keys must be specified with the key list, it is not used.
- For B+Tree files, the B+Tree index is not used.

See “dfred–Read a Logical Record” on page 134 and “DBRED–Read a Logical Record” on page 272 for examples of using Boolean logic on read operations.

Using Default-Key Key Lists

A default-key key list is used to make a set of default keys defined in the database definition (DBDEF) the active set of selection keys for read operations. The DBDEF keys specified can be either regular default keys or read-only default keys. Only the first key is filled in when using a default-key key list.

To use default keys on read operations, you must define a prototype LREC. The prototype LREC has the same layout as the LREC whose primary key is specified as the default key to be used. The prototype LREC must contain a search value in the fields corresponding to each key field in the default key.

The key list must be defined to have one key that must be set up with the specifications for default keys used on read operations:

- SW01ID1 #BIT2 must be on (set to 1). The #DF_KEYS (for assembler) or DF_KEYS (for C language) equate can be used to do this.
- SW01MSK must contain the primary key of a key that is defined in the database definition (DBDEF). This includes (but is not limited to) read-only default keys and LREC IDs. Read-only keys have LREC IDs X'01'–X'0F'. The remaining range of nonrestricted default keys (X'10' to X'EF') can be used on either read or add operations.
- SW01SEA must contain the base address of the prototype LREC containing the search values.

See “dfred–Read a Logical Record” on page 134 and “DBRED–Read a Logical Record” on page 272 for examples of using default keys on read operations.

Using Modification Key Lists

Global modifications of LRECs require a modification key list to be defined. This modification key list contains the same information as a standard selection key list, including primary keys and field displacements, lengths, and values. It also contains an operation code that indicates the operation to be performed on the selected LRECs. This operation code is specified in SW01ID1. See “DBSETK–Setting Up a Key in a Key List” on page 298 and “df_setkey–Setting Up a Key in a Key List” on page 150 for information about the values for these operation codes.

You must do the following to perform a global modification of LRECs:

1. If you want to specify a selection criteria to determine which LRECs will be modified, you must establish selection keys first. Do this by using a key list on a previous macro or function, or by using KEY*n* parameters on a previous macro.

2. Define a modification key list with the following information:
   - The displacement of the field in the LREC to be modified or used in the modification operation (SW01DIS)
   - The length of the field in the LREC to be modified, when appropriate (SW01LEN)
   - A 1-byte value (SW01MSK) or an address (SW01SEA) of the value to be used in the modification, depending on the modification operation being performed
   - The modification operation (SW01ID2).

   Note: Do ***not*** use a DBKEY macro or `dfkey` function with the modification key list.
3. Do one of the following:
   - Call the DBMOD macro with the ALL and MODLIST parameters specified and specify the base register of the modification key list.
   - Call one of the `dfmod_all` functions and specify a pointer to the modification key list.

Note: You can specify only 6 keys when using a modification key list.

See “dfmod–Perform or Indicate Logical Record Modifications” on page 117 and “DBMOD–Perform or Indicate Logical Record Modifications” on page 249 for examples of using global modification.

Sample Applications

This chapter contains examples of a typical application and provides the following:
- A description of the problem and solution
- The solution using the TPFDF assembler macros
- The solution using the TPFDF C functions.

Problem and Solution

Assume that you are in charge of the data processing department of a major airline. The director of marketing has created a new plan known as the Gold Club. Members of the Gold Club will have certain benefits not available to the general public, such as the use of a private lounge at most airports, the ability to pre-reserve seats, and the availability of special meals with 24 hours notice.

Members join the club by paying a membership fee, which covers one year's membership. Once a person is a member, he or she receives an additional month's free membership for every 1000 miles flown on the airline.

About 100 000 members are expected to join the club in its first year of operation. You must design a system to keep track of members, update their mileage credits from an existing departure control system, and update their membership expiration dates at regular intervals.

To see what the problem involves, write down all the relevant information for each member as follows:
- Member number
- Member surname
- Member initials
- Member address
- Meal preference
- Seat preference
- Payment method
- Expiration year
- Expiration month
- Miles credit.

In addition to setting up and maintaining the file, you must create a process that deducts 1000 miles from the current credit and adds one month to the expiration date whenever the mileage credit reaches 1000.

The easiest plan is to hold the information for each member in one fixed-length LREC with all the member LRECs being held in a fixed file.

Although this method could work adequately, it has the following disadvantages:
- The LRECs would have to be expanded as information is updated and changed.
- It is wasteful to have a large fixed file permanently allocated on DASD.

A better solution is to use basic indexing. In the detail file, each subfile holds a number of different LRECs, but all the LRECs in the subfile relate to one club member. Each subfile in the detail file is pointed to by an index LREC in the index file. Each index LREC also contains the member number as the index key.

© Copyright IBM Corp. 1997, 2001

The member information can be distributed over several variable-length LRECs. Different LREC types contain different information; for example:

Assembler LREC ID	C Language LREC ID	Data fields
X'80'	0x80	Member initials and surname.
X'90'	0x90	Member address. This is as many as six lines, separated by a delimiter. In our sample assembler application, an asterisk (*) is used as the delimiter. In our sample C language application, a null character (0\n) is used as the delimiter.
X'A0'	0xA0	Meal, seat, and payment method preferences.
X'B0'	0xB0	Mileage credit and current expiration date.

In addition, one subfile indexed using the dummy member number 9999999999, contains the following LRECs:

Assembler LREC ID	C Language LREC ID	Data fields
X'C0'	0xC0	Previously deleted member number.
X'D0'	0xD0	Next available membership number.

This subfile normally contains only one LREC. The LREC has one field (apart from the LREC ID), which is the next available member number (at startup, this is 0000000001).

However, when a member is deleted, the member number is returned to this subfile and can be reused. LRECs are organized in the subfile in available member number order, so any reusable member number LRECs are at the start of the subfile.

Some advantages of this method are:

- An application program only accesses LRECs that contain relevant information. Other information is in different LRECs.
- If an application program opens a subfile with hold, the held information only relates to one member. It is less likely that other applications need to read LRECs from the held subfile at the same time.
- This solution allows for easy addition to the subfiles. To include additional information about a member, you can add new LREC types without changing existing programs.

It might seem expensive in DASD storage to use 100 000 pool blocks for the detail file. However, you can choose any size from 381 bytes (L1 size) upward to suit your application. A reasonable size to allow for future expansion is 1055 bytes (L2). If the information about any member exceeds 1055 bytes, the TPFDF product creates overflow blocks automatically. You can choose the size of overflow blocks to be the same as the prime blocks or larger.

Note: To be very economical in the use of DASD, set the overflow blocks to a larger size (for example L4) and set bit 5 of SW00OP1. The TPFDF product then uses the smaller (prime block) size for overflow blocks unless they require the larger size.

Member File Definitions DSECT

A DSECT macro must be defined for each file that will be accessed by the application program. This is required for assembler application programs and C application programs.

Figure 9 shows the DSECT used to define the Gold Club member file.

```
         MACRO
&LABEL   IR00DF &REG=,&SUFFIX=,&ORG=,&ACPDB=
         GBLB  &IR00DF1      1ST TIME CALLED SWITCH
         COPY  DBGBL         COPY TPFDF GLOBAL DEFINITIONS
         COPY  DBLCL         COPY TPFDF LOCAL DEFINITIONS
&NAM     SETC  '             '    DOC NAME
&DATE    SETC  '06AUG90'     UPDATE DATE
&VERS    SETC  '00'          VERSION NUMBER
.*************************************************************
.*      DEFINITIONS FOR TPFDF                                *
.*************************************************************
&SW00WID SETC  'S0'          FILE ID
&SW00WRS SETC  'L2'          BLOCK SIZE
&SW00RBV SETC  '#TPFDBFF'    ALGORITHM
&SW02FIL SETC  'IR00DF'      FILE DSECT NAME
&SW00OP1 SETC  '00000000'    OPT BYTE1
&SW00OP2 SETC  '00000110'    OPT BYTE2
&SW00OP3 SETC  '00000000'    OPT BYTE3
&SW00TQK SETC  '15'          HIGHEST TLREC
.*************************************************************
         COPY  DBCOD         COPY DSECT DEFINITION FUNCTIONS
         AIF   ('&IR00DF1' EQ '1').NOT1ST
```

Figure 9. IR00DF–Member File Definitions DSECT (Part 1 of 5)

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                             *
*  DESCRIPTION OF IR00DF                                                      *
*                                                                             *
*  1. DATA AREA NAME                                                          *
*                                                                             *
*     GOLD CLUB MEMBER INFORMATION FILE                                       *
*                                                                             *
*  2. MEMBER NAME                                                             *
*                                                                             *
*     IR00DF                                                                  *
*                                                                             *
*  3. INVOCATION                                                              *
*                                                                             *
*     IR00DF REG=R4,                                                          *
*           (SUFFIX=X)                                                        *
*                                                                             *
*  4. GENERAL CONTENTS AND USAGE                                              *
*                                                                             *
*  4.1. ROLE IN SYSTEM                                                        *
*                                                                             *
*       DETAIL SUBFILE CONTAINING INFORMATION ON MEMBERS OF THE               *
*       GOLD CLUB SCHEME.  MEMBERS PAY FOR A ONE YEAR MEMBERSHIP              *
*       AND RECEIVE AN EXTRA MONTH'S MEMBERSHIP FOR EACH 1000 MILES           *
*       FLOWN WITH THE AIRLINE.                                               *
*                                                                             *
*  4.2. DATA LAYOUT                                                           *
*                                                                             *
*       STANDARD TPFDF FILE HEADER                                            *
*                                                                             *
*       PRIMARY KEY     USAGE                                                 *
*          80           MEMBER'S NAME                                         *
*          90           MEMBER'S ADDRESS                                      *
*          A0           MEAL, SEAT & PAYMENT PREFERENCES                      *
*          B0           MILEAGE CREDIT & EXPIRY DATE OF MEMBERSHIP            *
*          C0           PREVIOUSLY DELETED MEMBERSHIP NUMBER                  *
*          D0           NEXT MEMBERSHIP NUMBER TO GIVE OUT                    *
*                                                                             *
*  4.3. PROGRAMMING ASPECTS                                                   *
*                                                                             *
*  4.3.1. PROGRAMMING RESTRICTIONS                                            *
*                                                                             *
*         NONE.                                                               *
*                                                                             *
*  4.3.2. PROGRAMMING TECHNIQUES AND USAGE                                    *
*                                                                             *
*         STANDARD TPFDF LREC LOCATION TECHNIQUE USING:                       *
*         - PRIMARY KEY                                                       *
*                                                                             *
```

Figure 9. IR00DF–Member File Definitions DSECT (Part 2 of 5)

```
*  5. STORAGE FACTORS                                                  *
*                                                                      *
*  5.1. BLOCK SIZE                                                     *
*                                                                      *
*       1055 BYTES.                                                    *
*                                                                      *
*  5.2. FILE REQUIREMENTS                                              *
*                                                                      *
*       FIXED FILE INDEX (#TPFDB09) REFERENCING 1055-BYTE POOL FOR     *
*       DETAIL SUBFILE                                                 *
*                                                                      *
*  5.3. ACCESSING SCHEME                                               *
*                                                                      *
*       INDEXING THROUGH FILE IR01DF ACCESSED VIA ALG= STRING OF       *
*       THE MEMBERSHIP NUMBER.                                         *
*                                                                      *
*  6. DATA CONTROL                                                     *
*                                                                      *
*  6.1. CHAINING AND OVERFLOW                                          *
*                                                                      *
*       STANDARD TPFDF CHAINING.                                       *
*                                                                      *
*  6.2. DATA FIELD ADDRESSING                                          *
*                                                                      *
*       OFFSET WITHIN STANDARD TPFDF LREC.                             *
*                                                                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         EJECT
         AIF   ('&SW00WRS' EQ '').CHECKID
#IR00DFS EQU   &SW00WRS      BLOCK SIZE
.CHECKID AIF   ('&SW00WID' EQ '').NOT1ST
#IR00DFI EQU   C'&SW00WID'   FILE ID
.NOT1ST  ANOP
************************************************************************
*        STANDARD TPFDF HEADER                                         *
************************************************************************
IR00HDR&CG1 DS  CL16         STANDARD FILE HEADER
            DS  CL10         STANDARD TPFDF HEADER
IR00VAR&CG1 EQU *            START OF VARIABLE USER-AREA
IR00HDL&CG1 EQU IR00VAR&CG1-IR00HDR&CG1  HEADER-LENGTH UP TO IR00VAR
            ORG IR00HDR&CG1
IR00REC&CG1 DS  0CL1         1ST RECORD START (1=VARIABLE,ELSE SIZE)
IR00SIZ&CG1 DS  H            SIZE OF LOGICAL RECORD
IR00KEY&CG1 DS  X            LOGICAL RECORD IDENTIFIER
         AIF   ('&IR00DF1' EQ '1').KEYEQ  GO IF NOT FIRST ISSUE
```

Figure 9. IR00DF–Member File Definitions DSECT (Part 3 of 5)

```
*********************************************************************
*        EQUATE OF LOGICAL RECORD KEYS (KEY AND LENGTH)             *
*********************************************************************
.*                    USE KEY #IR00K80 IF ONLY ONE KEY
.*                    #IR00K00-#IR00K0F ARE RESERVED FOR TPFDF
.*                    #IR00KF0-#IR00KFF ARE RESERVED FOR TPFDF
#IR00K80 EQU   X'80'          PRIMARY KEY X'80'
#IR00K90 EQU   X'90'          PRIMARY KEY X'90'
#IR00KA0 EQU   X'A0'          PRIMARY KEY X'A0'
#IR00KB0 EQU   X'B0'          PRIMARY KEY X'B0'
#IR00KC0 EQU   X'C0'          PRIMARY KEY X'C0'
#IR00KD0 EQU   X'D0'          PRIMARY KEY X'D0'
.*
#IR00L80 EQU   IR00E80&CG1-IR00REC&CG1  LENGTH OF LOGICAL RECORD X'80'
#IR00L90 EQU   IR00E90&CG1-IR00REC&CG1  LENGTH OF LOGICAL RECORD X'90'
#IR00LA0 EQU   IR00EA0&CG1-IR00REC&CG1  LENGTH OF LOGICAL RECORD X'A0'
#IR00LB0 EQU   IR00EB0&CG1-IR00REC&CG1  LENGTH OF LOGICAL RECORD X'B0'
#IR00LC0 EQU   IR00EC0&CG1-IR00REC&CG1  LENGTH OF LOGICAL RECORD X'C0'
#IR00LD0 EQU   IR00ED0&CG1-IR00REC&CG1  LENGTH OF LOGICAL RECORD X'D0'
.*
&IR00SR1 SETB  (1)            INDICATE 1ST TIME THROUGH
.KEYEQ   ANOP
IR00ORG&CG1 EQU *             START OF LOGICAL RECORD DESCRIPTION
.*
*********************************************************************
*  MEMBER INITIALS AND SURNAME                                      *
*********************************************************************
.*
IR00NAM&CG1 DS  CL20          MEMBER'S SURNAME
IR00INT&CG1 DS  CL6           MEMBER'S INITIALS
IR00E80&CG1 EQU *             END OF LOGICAL RECORD WITH KEY X'80'
         ORG IR00ORG&CG1
.*
*********************************************************************
*  MEMBER ADDRESS                                                   *
*********************************************************************
.*
IR00ADR&CG1 DS  CL43          MEMBER'S ADDRESS
IR00E90&CG1 EQU *             END OF LOGICAL RECORD WITH KEY X'90'
         ORG IR00ORG&CG1
.*
*********************************************************************
*  MEAL, SEAT AND PAYMENT METHOD                                    *
*********************************************************************
.*
IR00SP0&CG1 DS  CL1           SPARE BYTE TO ALIGN ON HALFWORD BOUNDARY
IR00MPR&CG1 DS  H             MEAL PREFERENCE
IR00SPR&CG1 DS  H             SEAT REQUIREMENTS
IR00PAY&CG1 DS  H             PAYMENT METHOD
IR00EA0&CG1 EQU *             END OF LOGICAL RECORD WITH KEY X'A0'
         ORG IR00ORG&CG1
.*
```

Figure 9. IR00DF–Member File Definitions DSECT (Part 4 of 5)

```
*************************************************************************
*  MILEAGE CREDIT AND CURRENT EXPIRATION DATE                           *
*************************************************************************
.*
IR00SP1&CG1 DS  CL1          SPARE BYTE TO ALIGN ON HALFWORD BOUNDARY
IR00MLS&CG1 DS  F            CURRENT MILEAGE CREDIT
IR00EXY&CG1 DS  H            EXPIRATION YEAR
IR00EXM&CG1 DS  H            EXPIRATION MONTH
IR00EB0&CG1 EQU *            END OF LOGICAL RECORD WITH KEY X'B0'
         ORG IR00ORG&CG1
.*
*************************************************************************
*  REUSABLE MEMBER NUMBER                                               *
*************************************************************************
.*
IR00NUM&CG1 DS  CL10         REUSABLE MEMBER NUMBER
IR00EC0&CG1 EQU *            END OF LOGICAL RECORD WITH KEY X'C0'
         ORG IR00ORG&CG1
.*
*************************************************************************
*  CONSECUTIVE NUMBER                                                   *
*************************************************************************
.*
IR00NUC&CG1 DS  CL10         CONSECUTIVE MEMBER NUMBER CONTROL
IR00ED0&CG1 EQU *            END OF LOGICAL RECORD WITH KEY X'D0'
         ORG IR00ORG&CG1
.*
*************************************************************************
*  ALGORITHM DESCRIPTION                                                *
*************************************************************************
.*
         ORG   IR00REC&CG1
IR00@0BEG&CG1 EQU *                  PATH 0 DESCRIPTION
IR00@0NBR&CG1 DS  CL10               MEMBERSHIP NUMBER
IR00@0END&CG1 EQU *
.*
*************************************************************************
         AIF   (&BG1).MACEXIT        GO IF INTERNAL USAGE
&SYSECT  CSECT
         AIF   ('&REG' EQ '').MACEXIT  GO IF REG= NOT SPECIFIED
.GEUSING ANOP                        GENERATE USING
         USING &DSN,&REG
.MACEXIT ANOP
         SPACE 1
         MEND
```

Figure 9. IR00DF–Member File Definitions DSECT (Part 5 of 5)

Assembler Application Program Example

This section shows how you might code this example using the TPFDF macros.

Processing the Member File Using TPFDF Macros

The member file requires the following types of processing:

- **File maintenance**, which consists of adding, deleting, changing, and displaying information related to a member.

  Figure 10 on page 40 shows the file maintenance procedure.
- **Departure control interface**, which updates the member file with the number of miles flown after each departure and adjusts the membership expiration date.

Figure 11 on page 45 shows the departure control interface.

Note: No errors are issued from this program. Indicators are returned as shown.

- **Monthly maintenance**, which deletes records that have expired and adds the membership number to a subfile for reallocation.

  Figure 12 on page 47 shows the monthly maintenance procedure.

File Maintenance Program

```
         BEGIN NAME=SAM0,VERSION=00
         SPACE ,
*------------------------------------------------------------------------*
*        MEMBERSHIP FILE MAINTENANCE                                     *
*        Display a member's record : *member-nbr                         *
*        Delete a member's number  : Dmember-nbr                         *
*        Add a new member          : Amember-name                        *
*        Change members details    : Cmember-nbr/option/info             *
*                                    where option - A address            *
*                                                 - M meal preference    *
*                                                 - S seat preference    *
*                                                 - P payment method     *
*------------------------------------------------------------------------*
         MI0MI REG=R1        INPUT MESSAGE DSECT
         GLOBZ REGR=R2       BASE GLOBALS
         L     R1,EBCCR0     BASE OF INPUT MESSAGE BLOCK
*------------------------------------------------------------------------*
*        CHECK TO SEE IF SUBFILE IS TO BE OPENED WITH HOLD               *
*------------------------------------------------------------------------*
         #IF   CLI,MI0ACC,EQ,C'*'              DISPLAY REQUEST ?
            DBOPN REF=IR00DF,REG=R5,DETAC      NO NEED FOR HOLD
         #ELSE                                 MUST BE TYPE OF MODIFY
            DBOPN REF=IR00DF,REG=R5,HOLD,DETAC HOLD REQUIRED
         #EIF
*------------------------------------------------------------------------*
*        CHECK TO SEE IF MEMBER NUMBER IS TO BE DELETED                  *
*------------------------------------------------------------------------*
         #IF   CLI,MI0ACC,EQ,C'D'                DELETE REQUEST ?
            DBRED REF=IR00DF,REG=R5,ALG=MI0ACC+1, READ SUBFILE         X
               ERROR=SUBFILE-ERROR
            #IF   DBIDX,NO
               WTOPC TEXT='INVALID NUMBER',PREFIX=SAM0,NUM=001,LET=I
            #ELSE
*------------------------------------------------------------------------*
* DBCLS WITH RELFC HAS IMPLIED DEINDEX AND DBDEL,ALL.                    *
* SPECIFY THE REUSE PARAMETER TO USE THE SAME SW00SR SLOT TO             *
* UPDATE THE 'DELETED MEMBERS' SUBFILE                                   *
*------------------------------------------------------------------------*
               DBCLS REF=IR00DF,RELFC,NOKEY,REUSE
               DBOPN REF=IR00DF,REG=R5,HOLD,DETAC,ALG==C'9999999999'
*------------------------------------------------------------------------*
* BUILD NEW LREC FOR DELETED MEMBERSHIP NUMBER                           *
*------------------------------------------------------------------------*
               MVC   EBW000(L'IR00SIZ),=AL2(#IR00LC0)  SET UP SIZE
               MVI   EBW002,#IR00KC0                   PRIMARY KEY
               MVC   EBW003(L'IR00NUM),MI0ACC+1        DELETED NBR.
```

Figure 10. SAM0–File Maintenance Program (Part 1 of 5)

```
*--------------------------------------------------------------------*
* ADD NEW LREC - DEFAULT KEYS SPECIFIED IN DBDEF.                    *
*--------------------------------------------------------------------*
               DBADD REF=IR00DF,REG=R5,NEWLREC=EBW000,                X
               ERROR=SUBFILE-ERROR
            #EIF
          #ELIF CLI,MI0ACC,EQ,C'*'
            DBRED REF=IR00DF,REG=R5,ALG=MI0ACC+1,                     X
               ERROR=SUBFILE-ERROR
            #IF   DBIDX,NO
               WTOPC TEXT='INVALID NUMBER',PREFIX=SAM0,NUM=001,LET=I
            #ELSE
*--------------------------------------------------------------------*
* THIS IS JUST AN EXAMPLE OF THE USE OF DISPLAY, ITS USE IS NOT      *
* APPROPRIATE HERE AS SOME OF THE DATA IS NOT IN DISPLAYABLE FORMAT. *
* DBDSP AUTOMATICALLY DELETES THE SIZE FIELDS FROM THE DISPLAY.      *
*--------------------------------------------------------------------*
               DBDSP REF=IR00DF,STRIP==AL2(L'IR00KEY),NOKEY,          X
               ERROR=SUBFILE-ERROR
            #EIF
          #ELIF CLI,MI0ACC,EQ,C'A'
*--------------------------------------------------------------------*
* CHECK THE LENGTH OF THE INPUT MESSAGE DOESN'T EXCEDE MAX. ALLOWED  *
*--------------------------------------------------------------------*
            #IF   MI0CCT,GT,=AL2(L'IR00NAM+1)  INPUT MSG > MAXIMUM ?
               WTOPC TEXT='NAME EXCEEDS MAX.',PREFIX=SAM0,NUM=002,LET=E
            #ELSE
*--------------------------------------------------------------------*
* GET NEW MEMBERSHIP NUMBER                                          *
*--------------------------------------------------------------------*
               DBRED REF=IR00DF,REG=R5,ALG==C'9999999999',NOKEY,      X
               ERROR=SUBFILE-ERROR
               #IF   DBIDX,NO
*--------------------------------------------------------------------*
* DOESN'T EXIST, SO INITIALIZE THE FILE                              *
*--------------------------------------------------------------------*
                  DBADD REF=IR00DF,REG=R5,NEWLREC=SAM0INIT,           X
               ERROR=SUBFILE-ERROR
                  MVC   EBX000(L'IR00NUM),=C'0000000001'  NEW NBR.
               #ELSE
*--------------------------------------------------------------------*
* FILE EXISTS, SO CHECK TO SEE IF PREVIOUSLY DELETED NUMBER          *
*--------------------------------------------------------------------*
                  #IF   IR00KEY,EQ,#IR00KC0   PREVIOUSLY DELETED NBR.
                     MVC   EBX000(L'IR00NUM),IR00NUM
                     DBDEL REF=IR00DF,REG=R5,ERROR=SUBFILE-ERROR
                  #ELSE
```

Figure 10. SAM0–File Maintenance Program (Part 2 of 5)

```
*-------------------------------------------------------------------*
* THIS IS THE CONSECUTIVE LREC, SO MUST UPDATE IT,                  *
*-------------------------------------------------------------------*
                   MVC   EBX000(L'IR00NUC),IR00NUC
                   #PERF R7,UPDATE-NEXT-AVAIL-NUMBER
                #EIF
                DBCLS REF=IR00DF,REUSE
                DBOPN REF=IR00DF,REG=R5,HOLD
*-------------------------------------------------------------------*
* CREATE A NEW SUBFILE USING THE NEW MEMBER NBR. AS ALG= STRING.    *
*-------------------------------------------------------------------*
                DBCRE REF=IR00DF,INDEX,ALG=EBX000,                  X
             ERROR=NEW-ERROR
*-------------------------------------------------------------------*
* BUILD THE NAME LREC FROM THE INPUT MSG.                           *
*-------------------------------------------------------------------*
                LH    R4,MI0CCT        GET INPUT MESSAGE LENGTH
                #STPR R4,-2            ADJUST FOR 'A' PART & EX
                EX    R4,SAM0MOV1      START TO BUILD NEW LREC
                #STPR R4,L'IR00KEY+l'IR00SIZ  SIZ + PRIMARY KEY
                STH   R4,EBW000        STORE SIZE
                MVI   EBW002,#IR00L80  SET UP PRIMARY KEY
*-------------------------------------------------------------------*
* ADD THE NAME LREC AND DUMMY LRECS FOR THE OTHER INFORMATION.      *
*-------------------------------------------------------------------*
                DBADD REF=IR00DF,REG=R5,NEWLREC=EBW000,             X
             ERROR=NEW-ERROR
                DBADD REF=IR00DF,REG=R5,NEWLREC=SAM0ADDR,           X
             ERROR=NEW-ERROR
                DBADD REF=IR00DF,REG=R5,NEWLREC=SAM0MEAL,           X
             ERROR=NEW-ERROR
                DBADD REF=IR00DF,REG=R5,NEWLREC=EBW000,             X
             ERROR=NEW-ERROR
*-------------------------------------------------------------------*
* UPDATE THE EXPIRATION DATE FOR THE NEW MEMBER, TODAY + A YEAR.    *
*-------------------------------------------------------------------*
                LH    R4,@YEAR
                #STPR R4,1
                STH   R4,IR00EXY
                MVC   IR00EXM,@MONTH
                DBMOD REF=IR00DF
                MVC   EBW000(L'SAM0MSG1),SAM0MSG1  SKELETON MSG
                MVC   EBW000+21(L'IR00NUM),EBX000  MOVE IN NEW NBR
                WTOPC TEXTA=EBW000,PREFIX=SAM0,NUM=006,LET=I
             #EIF
          #EIF
```

Figure 10. SAM0–File Maintenance Program (Part 3 of 5)

```
*--------------------------------------------------------------------*
*     MODIFY REQUEST                                                 *
*--------------------------------------------------------------------*
         #ELIF CLI,MI0ACC,EQ,C'C',AND,   CHANGE REQUEST ?
         #     CLI,MI0ACC+11,EQ,C'/'
            #IF   CLI,MI0ACC+12,EQ,C'A',AND,       ADDRESS ?
            #     MI0CCT,GT,=AL2(L'IR00ADR+L'IR00NUM+4) INPUT OK ?
                  DBRED REF=IR00DF,ALG=MI0ACC+1,KEY1=(PKY=#IR00K90),   X
            REG=R5,ERROR=SUBFILE-ERROR
                  LH    R4,MI0CCT     LOAD LENGTH OF INPUT
                  SH    R4,=AL2(L'IR00NUM+4)    ADJUST FOR EX & NUMBER
                  EX    R4,SAM0MOV2             MOVE
                  #STPR R4,l'IR00SIZ+l'IR00KEY
                  STH   R4,IR00SIZ
                  MVI   IR00KEY,#IR00K90
                  DBMOD REF=IR00DF,REG=R5
            #ELIF CLI,MI0ACC+12,EQ,C'M',OR,       MEAL ?
            #     CLI,MI0ACC+12,EQ,C'S',OR,       SEAT ?
            #     CLI,MI0ACC+12,EQ,C'P',ANDIF,    PAYMENT ?
            #     MI0CCT,GT,=AL2(L'IR00MPR+L'IR00NUM+4) INPUT OK ?
               DBRED REF=IR00DF,ALG=MI0ACC+1,KEY1=(PKY=#IR00KA0),      X
               REG=R5,ERROR=SUBFILE-ERROR
               LH    R4,MI0ACC+15       GET NEW INFORMATION
               #IF   MI0ACC+12,EQ,C'M'        MEAL ?
                  ST    R4,IR00MPR            STORE IT IN CORRECT FIELD
               #ELIF MI0ACC+12,EQ,C'S'        SEAT ?
                  ST    R4,IR00SPR            STORE IT IN CORRECT FIELD
               #ELIF MI0ACC+12,EQ,C'P'        PAYMENT ?
                  ST    R4,IR00PAY            STORE IT IN CORRECT FIELD
               #EIF
               DBMOD REF=IR00DF,REG=R5        UPDATE DATABASE
               WTOPC TEXT='DETAILS MODIFIED',PREFIX=SAM0,NUM=005,LET=E
            #EIF
            WTOPC TEXT='INVALID INPUT',PREFIX=SAM0,NUM=005,LET=E
         #EIF
         #LOCA EXIT
         DBCLS REF=IR00DF,RELEASE      CLOSE SUBFILE & RELEASE SW00SR
         EXITC ,
         SPACE ,
*--------------------------------------------------------------------*
* SUBROUTINE TO INCREMENT CONSECUTIVE MEMBER RECORD BY 1 (DATA PACKED)*
*--------------------------------------------------------------------*
         SPACE ,
         #SUBR UPDATE-NEXT-AVAIL-NUMBER
            PACK EBX010(L'IR00NUC),EBX000(L'IR00NUC)   PACK DATA
            AP   EBX010(L'IR00NUC),=P'1'               ADD '1'
            UNPK EBX000(L'IR00NUC),EBX010(L'IR00NUC)   UNPACK DATA
            OI   EBX009,X'F0'                          CLEAR SIGN BIT
            #IF  EBX000(L'IR00NUC),EQ,=C'9999999999'   NO MORE NBRS
               #GOTO DATABASE-FULL
            #EIF
            MVC  IR00NUC,EBX000         UPDATE LREC
            DBMOD REF=IR00DF,REG=R5     UPDATE DATABASE
         #ESUB
```

Figure 10. SAM0–File Maintenance Program (Part 4 of 5)

```
*--------------------------------------------------------------------*
* ERROR HANDLING                                                     *
*--------------------------------------------------------------------*
         #LOCA SUBFILE-ERROR
         WTOPC TEXT='ERROR IN DETAIL SUBFILE',PREFIX=SAM0,NUM=006,LET=E
         #GOTO EXIT
         #LOCA NEW-ERROR
         WTOPC TEXT='ERROR IN NEW SUBFILE',PREFIX=SAM0,NUM=007,LET=E
         #GOTO EXIT
         #LOCA DATABASE-FULL
         WTOPC TEXT='DATABASE FULL',PREFIX=SAM0,NUM=006,LET=E
*--------------------------------------------------------------------*
* EXECUTABLE MOVE INSTRUCTIONS                                       *
*--------------------------------------------------------------------*
SAM0MOV1 MVC   EBW003(0),MI0ACC+1      EXECUTABLE MOVE INSTR.
SAM0MOV2 MVC   IR00ADR(0),MI0ACC+13    EXECUTABLE MOVE INSTR.
         SPACE ,
*--------------------------------------------------------------------*
* LREC TO INITIALIZE NUMBERS SUBFILE, FIRST NUMBER TO BE ADDED IS    *
* '2' AS WILL ONLY BE INITIALIZED WHEN NBR '1' IS ALLOCATED.         *
*--------------------------------------------------------------------*
         SPACE ,
SAM0INIT DC    AL2(#IR00LD0),AL1(#IR00KD0),C'0000000002'
         SPACE ,
*--------------------------------------------------------------------*
* DUMMY RECORD TO INITIALIZE NEW MEMBER'S SUBFILE                    *
*--------------------------------------------------------------------*
         SPACE ,
SAM0ADDR DC    AL2(#IR00L90),AL1(#IR00K90),CL43' '
SAM0MEAL DC    AL2(#IR00LA0),AL1(#IR00KA0),CL7' '
SAM0INFO DC    AL2(#IR00LB0),AL1(#IR00KB0),C' ',XL6'00'
         SPACE ,
SAM0MSG1 DC    AL1(SAM0END1-SAM0MSG1)
         DC    C'NEW MEMBER CREATED - ..........'
SAM0END1 EQU   *
         SPACE ,
         LTORG
         FINIS SAM0
         END
```

Figure 10. SAM0–File Maintenance Program (Part 5 of 5)

Departure Control Interface Program

```
         BEGIN NAME=SAM1,VERSION=00
         SPACE ,
*-------------------------------------------------------------------*
*        DEPARTURE CONTROL INTERFACE - UPDATES THE MILEAGE FLOWN AND  *
*        ADJUSTS THE EXPIRATION DATE OF THE MEMBERSHIP BASED ON AN    *
*        EXTRA MONTH'S MEMBERSHIP FOR EACH 1000 MILES FLOWN.          *
*        NO ERRORS ARE ISSUED FROM THIS PROGRAM, INDICATORS ARE       *
*        RETURNED AS SHOWN BELOW.                                     *
*-------------------------------------------------------------------*
*        INPUT CONDITIONS                                             *
*            EBW000(L'10) - MEMBERSHIP NUMBER                         *
*            EBW010(L'2 ) - MILEAGE TO CREDIT                         *
*                                                                     *
*        OUTPUT CONDITIONS                                            *
*             EBSW01 X'80'- NO SUBFILE FOR MEMBER NUMBER SPECIFIED    *
*                    X'40'- NO MILEAGE LREC IN SUBFILE                *
*                    X'20'- SERIOUS ERROR ON READ                     *
*                    X'00'- EVERYTHING OK                             *
*-------------------------------------------------------------------*
         SPACE ,
*-------------------------------------------------------------------*
*        OPEN SUBFILE USING MEMBER-NBR. AS ALG= STRING                *
*-------------------------------------------------------------------*
         SPACE ,
         DBOPN REF=IR00DF,REG=R5,HOLD
         SPACE ,
```

Figure 11. SAM–Departure Control Interface (Part 1 of 2)

```
*-------------------------------------------------------------------*
*        SEARCH FOR MILEAGE LREC BASED ON ITS UNIQUE PRIMARY KEY     *
*-------------------------------------------------------------------*
         SPACE ,
         DBRED REF=IR00DF,REG=R5,ALG=EBW000,                        X
               KEY1=(PKY=#IR00KB0),                                 X
               ERROR=READ-ERROR
         #IF   DBIDX,NO              DETAIL SUBFILE INDEXED
            MVI   EBSW01,X'80'       INDICATE ERROR
         #ELIF DBFOUND,NO            MILEAGE LREC NOT FOUND
            MVI   EBSW01,X'40'       INDICATE ERROR
         #ELSE
            LH    R6,EBW010          MILEAGE TO CREDIT
            A     R6,IR00MLS         ADD TO EXISTING MILEAGE
            SRDA  R6,32(0)           SET UP EVEN-ODD REG. FOR D
            D     R6,=H'1000'        CALC THE INC. OF EXPIRATION DATE
            AH    R7,IR00EXM         ADD EXTRA MONTHS TO EXP. MONTH
            #IF   R7,GT,=H'12'       MORE THAN 12 MONTHS
               #DO   WHILE=(R7,GT,=H'12')  WHILE MORE THAN 12 MONTHS
                  #STPR R7,-12            DELETE A YEAR
                  #STPH R6,1,IR00EXY      ADD A YEAR TO EXPIRATION DATE
               #EDO
            #EIF
            STH   R7,IR00EXM         STORE NBR OF EXPIRATION MONTHS
            DBMOD REF=IR00DF,REG=R5  UPDATE DATABASE
            MVI   EBSW01,X'00'       EVERYTHING OK
         #EIF
         #LOCA EXIT
         DBCLS REF=IR00DF,RELEASE    CLOSE SUBFILE, RELEASE SW00SR SLOT
         BACKC ,                     RETURN TO CALLER
*-------------------------------------------------------------------*
*        ERROR HANDLING                                             *
*-------------------------------------------------------------------*
         #LOCA READ-ERROR
         MVI   EBSW01,X'20'          INDICATE ERROR
         #GOTO EXIT                  CLOSE FILE & RETURN TO CALLER
         LTORG
         FINIS SAM1
         END
```

Figure 11. SAM–Departure Control Interface (Part 2 of 2)

Monthly Maintenance Program

```
         BEGIN NAME=SAM2,VERSION=00
         SPACE ,
*------------------------------------------------------------------*
*        MONTHLY MAINTENANCE PROGRAM, DELETES MEMBERS RECORDS WHICH  *
*        HAVE EXPIRED  AND ADDS THE MEMBERSHIP NUMBER TO A SUBFILE   *
*        SO THAT THEY CAN BE RE-ISSUED.                              *
*------------------------------------------------------------------*
         GLOBZ REGR=R2               BASE GLOBALS
         #IF   CLI,@DAY,NE,X'01'     NOT FIRST DAY OF MONTH
            WTOPC TEST='NOT FIRST OF MONTH - UNABLE TO PROCESS',      X
               PREFIX=SAM2,NUM=001,LET=I
         #ELSE
*------------------------------------------------------------------*
* OPEN DETAIL SUBFILE FOR FULLFILE PROCESSING (IR00DF) AND OPEN THE  *
* SUBFILE WHICH CONTAINS DELETED NUMBERS (ALG==C'9999999999').       *
*------------------------------------------------------------------*
            DBOPN REF=IR00DF,HOLD,REG=R5
            DBOPN REF=IR00DFA,HOLD,REG=R6,                            X
               ALG==C'9999999999',SUFFIX=A
            #DO   INF                INFINITE LOOP - EXIT WHEN EOF
            SPACE ,
*------------------------------------------------------------------*
*   READ EACH EXPIRATION DATE LREC, SAVING MEMBER NUMBER FROM INDEX  *
*   FILE IN EBW000 USING AREA PARAMETER (SETUP IN DBDEF).            *
*------------------------------------------------------------------*
               SPACE ,
               DBRED REF=IR00DF,REG=R5,FULLFILE,                      X
               AREA=EBW003,                                           X
               KEY1=(PKY=#IR00KB0),                                   X
               ERROR=READ-ERROR
               #DOEX DBEOF,YES             EXIT LOOP IF END OF FILE
               #IF   IR00EXM,LT,@MONTH,AND, EXP. MONTH < CURRENT MONTH
               #     IR00EXY,LE,@YEAR,ORIF,  & EXP. YEAR <= CURR. YEAR
               #     IR00EXY,LT,@YEAR       EXP. YEAR < CURRENT YEAR
                  #PERF R7,UPDATE-FREE-SLOT    EXPIRED MEMBERSHIP
                  SPACE ,
*------------------------------------------------------------------*
* DELETE ALL LOGICAL RECORDS IN DETAIL SUBFILE, DE-INDEX IS AUTOMATIC *
*------------------------------------------------------------------*
                  SPACE ,
                  DBDEL REF=IR00DF,ALL,NOKEY,ALG=EBW003
               #EIF
            #EDO
            WTOPC TEXT='PROCESSING COMPLETED',PREFIX=SAM2,NUM=002,     X
               LET=I
*------------------------------------------------------------------*
* CLOSE BOTH FILES AND RELEASE SW00SR SLOTS                         *
*------------------------------------------------------------------*
            #LOCA EXIT
            DBCLS REF=IR00DF,RELEASE
            DBCLS REF=IR00DFA,RELEASE
         #EIF
         EXITC ,
         SPACE ,
```

Figure 12. SAM2–Monthly Maintenance Program (Part 1 of 2)

```
*-------------------------------------------------------------------*
* SUBROUTINE TO UPDATE SUBFILE WITH NUMBERS TO REUSE                *
*-------------------------------------------------------------------*
         #SUBR UPDATE-FREE-SLOT
            MVC   EBW000(L'IR00SIZ+L'IR00KEY),SAM2INFO  LREC SIZE + KEY
            SPACE ,
*-------------------------------------------------------------------*
* ADD LREC CONTAINING DELETED MEMEBER NBR. - ORGANIZATION OF FILE   *
* DETERMINED BY THE DEFAULT KEYS DEFINED IN DBDEF.                  *
*-------------------------------------------------------------------*
            SPACE ,
            DBADD REF=IR00DFA,NEWLREC=EBW000,                        X
               ERROR=ADD-ERROR
         #ESUB ,
         SPACE ,
*-------------------------------------------------------------------*
* ERROR HANDLING                                                    *
*-------------------------------------------------------------------*
         SPACE ,
         #LOCA ADD-ERROR
         SERRC R,DF0000        ISSUE DUMP & SEND MSG
         WTOPC TEXT='JOB ABORTED - ADD ERROR ON FILE IR00DF',         X
               PREFIX=SAM2,NUM=003,LET=E
         #GOTO EXIT            GO & CLOSE FILES
         #LOCA READ-ERROR
         SERRC R,DF0001        ISSUE DUMP & SEND MSG
         WTOPC TEXT='JOB ABORTED - READ-ERROR ON FILE IR00DF',        X
               PREFIX=SAM2,NUM=004,LET=E
         #GOTO EXIT            GO & CLOSE FILES
         SPACE ,
SAM2INFO DC    AL2(L'IR00LC0),AL1(#IR00KC0)   LENGTH + PRIMARY KEY
         LTORG
         FINIS SAM2
         END
```

Figure 12. SAM2–Monthly Maintenance Program (Part 2 of 2)

C Language Application Program Example

This section shows how you might code this example using the TPFDF C language functions.

You must ensure that there is a structure defined for each file that the C language application program will access. Figure 13 on page 49 shows the structure of the member file used in this example.

```
1  /*--------------------------------------------------------------------*
2   *      Passenger Record Structure Declaration                        *
3   *--------------------------------------------------------------------*/
4  #define _IR00DFI        "S0"            /* File ID */
5
6  #define _IR00K80        0x80            /* logical record keys */
7  #define _IR00K90        0x90
8  #define _IR00KA0        0xA0
9  #define _IR00KB0        0xB0
10 #define _IR00KC0        0xC0
11 #define _IR00KD0        0xD0
12
13 #define _IR00L80        sizeof(struct ir00psgr) /* passenger name     */
14 #define _IR00L90        sizeof(struct ir00addr) /* passenger address  */
15 #define _IR00LA0        sizeof(struct ir00info) /* meal, seat, pay    */
16 #define _IR00LB0        sizeof(struct ir00stat) /* membership status  */
17 #define _IR00LC0        sizeof(struct ir00rnum) /* reusable number    */
18 #define _IR00LD0        sizeof(struct ir00cnum) /* consecutive number */
19
```

Figure 13. ir00df.h–Passenger Record Structure Declaration (Part 1 of 2)

```
20  struct ir00df
21 {
22   short   ir00siz;                    /* LREC size   */
23   dft_pky ir00key;                    /* primary key */
24   union
25   {
26     struct ir00psgr
27     {
28       char  ir00nam[20];              /* surname */
29       char  ir00int[6];               /* initials */
30     } psgr;
31
32     struct ir00addr
33     {
34       char ir00adr[43];               /* address */
35     } addr;
36
37     struct ir00info
38     {
39       char  ir00sp0;                  /* spare for alignment */
40       char  ir00mpr[2];               /* meal preference     */
41       char  ir00spr[2];               /* seat preference     */
42       char  ir00pay[2];               /* payment method      */
43     } info;
44
45     struct ir00stat
46     {
47       char  ir00sp1;                  /* spare for alignment    */
48       int   ir00mls;                  /* current mileage credit */
49       short ir00exy;                  /* expiration year        */
50       short ir00exm;                  /* expiration month       */
51     } stat;
52
53     struct ir00rnum
54     {
55       char  ir00num[10];              /* reusable member number */
56     } rnum;
57
58     struct ir00cnum
59     {
60       char  ir00nuc[10];              /* consecutive member number */
61     } cnum;
62   } lrec;
63 };
```

Figure 13. ir00df.h–Passenger Record Structure Declaration (Part 2 of 2)

Figure 14 on page 51 shows how you can redefine TPFDF functions so that you can code them more simply in the application program.

Note: This type of header is not required; you can code the TPFDF functions directly in your application program if you prefer.

```
/*--------------------------------------------------------------------*
 *       MEMBER FILE DEFINITIONS HEADER                               *
 *                                                                    *
 *       This file contains definitions to support use of the         *
 *       Gold Club Passenger (member) file.                           *
 *--------------------------------------------------------------------*/
#include "ir00df.h"

#define HOLD            DFOPN_HOLD
#define NOHOLD          DFOPN_NOHOLD
#define RELFC           DFCLS_RELFC
#define REUSE           DFCLS_REUSE

#define open_psgr(opts)         dfopn("PSGRFILE",_IR00DFI,opts)
#define create_psgr(file,alg)   (void) dfcre_alg(file,0,alg)
#define read_psgr(file,alg)     dfred_acc(file,DFRED_ALG,0,alg)
#define add_psgr(file,rcd)      (void) dfadd(file,DFADD_NEWLREC,0,rcd)
#define replace_psgr(file,rcd)  (void) dfrep(file,DFREP_NEWLREC,rcd)
#define modify_psgr(file)       (void) dfmod(file)
#define close_psgr(file,opts)   dfcls(file,opts)
#define display_psgr(file)      dfdsp_str(file,DFDSP_NOKEY, \
                                    offsetof(struct ir00df,ir00key))
#define open_del_nbr(opts)      dfopn_acc("DELNBRS ",_IR00DFI,DFOPN_ALG, \
                                          DFOPN_DETAC|opts, \
                                          "9999999999")
#define read_del_nbr(file)      dfred_acc(file,DFRED_ALG,DFRED_NOKEY, \
                                         "9999999999")
#define add_del_nbr(file,rcd)   (void) dfadd(file,DFADD_NEWLREC,0,rcd)
#define delete_del_nbr(file)    (void) dfdel(file,0)
#define modify_del_nbr(file)    (void) dfmod(file)
#define close_del_nbr(file)     dfcls(file,0)

#define MEM_NUM_SIZE    member_size(struct ir00df,lrec.rnum.ir00num)
#define MAX_NAME_SIZE   member_size(struct ir00df,lrec.psgr.ir00nam)
#define MAX_ADDR_SIZE   member_size(struct ir00df,lrec.addr.ir00adr)
#define MEAL_SIZE       member_size(struct ir00df,lrec.info.ir00mpr)
#define SEAT_SIZE       member_size(struct ir00df,lrec.info.ir00spr)
#define PMNT_SIZE       member_size(struct ir00df,lrec.info.ir00pay)
```

Figure 14. psgr.h–Member File Definitions Header

Processing the Member File Using TPFDF C Functions

The member file requires the following types of processing:

- **File maintenance**, which consists of adding, deleting, changing, and displaying information related to a member.

  Figure 15 on page 52 shows the file maintenance procedure.
- **Departure control interface**, which updates the member file with the number of miles flown after each departure and adjusts the membership expiration date.

  Figure 16 on page 63 shows the departure control interface.
- **Monthly maintenance**, which deletes records that have expired and adds the membership number to a subfile for reallocation.

  Figure 17 on page 65 shows the monthly maintenance procedure.

File Maintenance Program

```
 1  /*--------------------------------------------------------------------*
 2   *       MEMBERSHIP FILE MAINTENANCE                                  *
 3   *       Display a member's record : *member-nbr                      *
 4   *       Delete a member's number  : Dmember-nbr                      *
 5   *       Add a new member          : Amember-name                     *
 6   *       Change members details    : Cmember-nbr/Xinfo                *
 7   *                                   where X is:  - A address         *
 8   *                                                - M meal preference *
 9   *                                                - S seat preference *
10   *                                                - P payment method  *
11   *--------------------------------------------------------------------*/
12  #include <stdlib.h>     /* standard include files   */
12A #include <time.h>
13  #include <cdf.h>        /* TPFDF include files      */
13A #include <psgr.h>
14
15  #define op_msg(prefix,text) \
16   ((void) time(&ltime), \
17    printf("%s %s %s\n",prefix,ctime(&ltime),text))
18
19  #define read_psgr_lrec(file,alg,pky) \
20                 (df_nbrkeys(&psgr_keys,1), \
21                  psgr_pky = pky, \
22                  df_setkey(&psgr_keys,1, \
23                            offsetof(struct ir00df,ir00key), \
24                            1,DF_EQ,&psgr_pky,0,DF_UPORG), \
25                  read_psgr(file,alg))
26
27  static time_t ltime;            /* used for time conversion in op_msg */
28
29  static dft_kyl psgr_keys;       /* passenger file key area            */
30  static dft_pky psgr_pky;        /* passenger file primary key         */
31
32  static char in_msg[128];        /* activation message buffer          */
33  static dft_fil *psgr_file;      /* passenger file pointer             */
34  static dft_fil *del_nbr_file;   /* 'deleted numbers' subfile          */
35
36  static void action_delete(void);
37  static void action_display(void);
38  static void action_add(void);
39  static void action_change(void);
40  static void subfile_error(void);
41  static void new_error(void);
42  static void data_base_full(void);
43  static void update_next_available_number(struct ir00df *);
44  static void change_address(void);
45  static void change_info(char);
46
```

Figure 15. sam0.c–File Maintenance Program (Part 1 of 11)

```
/*--------------------------------------------------------------------*
 *      Open the Passenger File and Determine the Required Action     *
 *--------------------------------------------------------------------*/
void samp(void)
{
  /* local variables */
  unsigned msg_len;

  /* Read the activation message and remove any EOM character. */
  (void) gets(in_msg);
  msg_len = strlen(in_msg);

  if (in_msg[msg_len]== _EOM)
    in_msg[msg_len]= 0x00;

  /* If it is not a display request, the passenger */
  /* file must be open with the HOLD option.       */
  psgr_file = open_psgr(in_msg[0]== '*' ? NOHOLD : HOLD);

  /* Determine the required action. */
  switch (in_msg[0])
  {
    /* Delete Passenger Subfile. */
    case 'D':
      action_delete();
      break;

    /* Display Passenger Subfile. */
    case '*':
      action_display();
      break;

    /* Add New Passenger Subfile. */
    case 'A':
      action_add();
      break;

    /* Change Passenger Information. */
    case 'C':
      action_change();
      break;

    /* Invalid Input. */
    default:
      op_msg("SAM005E","UNRECOGNIZED ACTION CODE");
      break;
  }

  /* Close the passenger file, use the RELFC option */
  /* if the action was delete (dfcls with RELFC     */
  /* has implied de-index and DBDEL_ALL).           */
  close_psgr(psgr_file,in_msg[0]== 'D' ? RELFC : 0);

  exit(0);
}

```

Figure 15. sam0.c–File Maintenance Program (Part 2 of 11)

```
103 /*--------------------------------------------------------------------*
104  *      Read Passenger Record and Verify its Existence                *
105  *                                                                    *
106  *              Returns pointer to record if subfile found,           *
107  *              otherwise returns NULL                                *
108  *--------------------------------------------------------------------*/
109 static struct ir00df *verify_number()
110 {
111   /* local variables */
112   static struct ir00df *psgr_lrec;      /* passenger record         */
113
114   /* Read a record from the passenger subfile referenced by the     */
115   /* member number supplied in the input message.  If the subfile   */
116   /* does not exist, then the member number is incorrect.           */
117   psgr_lrec = read_psgr(psgr_file,&in_msg[1]);
118
119   if (DF_ERX(psgr_file))                /* if error from read       */
120     subfile_error();                    /* doesn't return           */
121
122   if (DF_TEST(psgr_file,DFC_ALG))       /* if subfile doesn't exist */
123     op_msg("SAM001E","INVALID NUMBER"); /* send message to operator */
124
125   return (psgr_lrec);
126 }
127
128 /*--------------------------------------------------------------------*
129  *      Delete Passenger Subfile                                      *
130  *--------------------------------------------------------------------*/
131 static void action_delete(void)
132 {
133   /* Verify that the supplied member number exists. */
134   if (verify_number())
135   {
136     /* local variables */
137                 /* partially initialized 'deleted number' LREC */
138     struct ir00df del_nbr_lrec = { _IR00LC0, _IR00KC0 };
139
140     /* Open the 'deleted numbers' subfile. */
141     del_nbr_file = open_del_nbr(HOLD);
142
143     /* Build a new LREC for the deleted membership number */
144     /* and add it to the 'deleted numbers' subfile.       */
145     (void) strncpy(del_nbr_lrec.lrec.rnum.ir00num,
146                    &in_msg[1],
147                    sizeof(del_nbr_lrec.lrec.rnum.ir00num));
148     add_del_nbr(del_nbr_file,&del_nbr_lrec);
149
150     if (DF_ER(del_nbr_file))            /* if error from add */
151       subfile_error();                  /* doesn't return    */
152   }
153   return;
154 }
155
```

Figure 15. sam0.c–File Maintenance Program (Part 3 of 11)

```
/*--------------------------------------------------------------------*
 *      Display Passenger Subfile                                     *
 *--------------------------------------------------------------------*/
static void action_display(void)
{
  /* Verify that the supplied member number exists. */
  if (verify_number())
  {
    /* This is just an example of the use of dfdsp.  Its use is not */
    /* appropriate here as some of the data is not in displayable   */
    /* format.  dfdsp automatically skips the size field.           */
    display_psgr(psgr_file);

    if (DF_ER(psgr_file))               /* if error from display    */
      subfile_error();                  /* doesn't return           */
  }
  return;
}

/*--------------------------------------------------------------------*
 *      Add New Passenger Subfile                                     *
 *--------------------------------------------------------------------*/
static void action_add(void)
{
  /* local variables */

  /*--------------------------------------------------------------------*
   * LREC to initialize the 'deleted numbers' subfile.  The first     *
   * number to be added is 2, since this subfile will only be         *
   * initialized when member number 1 is issued.                      *
   *--------------------------------------------------------------------*/
  struct ir00df del_nbr_init = { _IR00LD0, _IR00KD0 };

  struct ir00df new_psgr_lrec;          /*build area for psgr LREC  */
  struct tm *tp;                        /*current time structure    */
  char next_num[MEM_NUM_SIZE+1];        /*next member number to use */
  const char done_msg[]= "NEW MEMBER FILECREATED - ";
  char msg_text[MEM_NUM_SIZE+sizeof(done_msg)+1];

  /* Verify the length of the input message. */
  if (strlen(&in_msg[1]) > MAX_NAME_SIZE)
    op_msg("SAM002E","NAME TOO LONG");  /* send message to operator */
  else

```

Figure 15. sam0.c–File Maintenance Program (Part 4 of 11)

```
 /* Message length is OK, so continue. */
{
  /* local variables */
  struct ir00df *del_nbr_rcd;

  /* Get new membership number. */
  del_nbr_file = open_del_nbr(HOLD);
  del_nbr_rcd = read_del_nbr(del_nbr_file);

  if (DF_ERX(del_nbr_file))           /* if error from read */
    subfile_error();                  /* doesn't return     */

  if (DF_TEST(del_nbr_file,DFC_ALG))  /* if subfile doesn't exist */
  {
    /* The 'deleted numbers' subfile doesn't exist, so create it. */
    (void) strncpy(del_nbr_init.lrec.rnum.ir00num,"0000000002",10);
    add_del_nbr(del_nbr_file,&del_nbr_init);

    if (DF_ER(del_nbr_file))          /* if error from read */
      subfile_error();                /* doesn't return     */

    (void) strcpy(next_num,"0000000001"); /* new number */
  }
  else
  {
    /* 'deleted numbers' subfile exists, */
    /* so find an available number       */

    /* If the found record is a deleted number, use the number */
    /* and delete the record from the 'deleted numbers' file.  */
    if (del_nbr_rcd->ir00key == _IR00KC0)
    {
      (void) strncpy(next_num,
                     del_nbr_rcd->lrec.rnum.ir00num,
                     MEM_NUM_SIZE);
      delete_del_nbr(del_nbr_file);
    }
    else
    {
      /* The found record is a consecutive number, */
      /* so use it and increment it in the file.   */
      (void) strncpy(next_num,
                     del_nbr_rcd->lrec.cnum.ir00nuc,
                     MEM_NUM_SIZE);
      update_next_available_number(del_nbr_rcd);
      modify_del_nbr(del_nbr_file);
    }
  }

```

Figure 15. sam0.c–File Maintenance Program (Part 5 of 11)

```
249     /* Close the 'deleted numbers' subfile. */
250     close_del_nbr(del_nbr_file);
251
252     /* Create a new subfile using the new member */
253     /* number as the algorithm argument.         */
254     close_psgr(psgr_file,REUSE);
255     psgr_file = open_psgr(HOLD);
256     create_psgr(psgr_file,next_num);
257
258     if (DF_ER(psgr_file))               /* if error from create */
259       new_error();                      /* doesn't return       */
260
261     /* Build the name LREC from the input */
262     /* message and add it to the file.    */
263     new_psgr_lrec.ir00siz = (short) (strlen(&in_msg[1]) +
264                                      sizeof(new_psgr_lrec.ir00siz) +
265                                      sizeof(new_psgr_lrec.ir00key));
266     new_psgr_lrec.ir00key = _IR00K80;
267     (void) strncpy(new_psgr_lrec.lrec.psgr.ir00nam,
268                    &in_msg[1],
269                    MAX_NAME_SIZE);
270
271     add_psgr(psgr_file,&new_psgr_lrec);
272
273     if (DF_ER(psgr_file))
274       new_error();
275
276     /* Build the status LREC, which contains the current */
277     /* mileage credit (0), and the expiration date       */
278     /* (today + one year).                               */
279     new_psgr_lrec.ir00siz = _IR00LB0;
280     new_psgr_lrec.ir00key = _IR00KB0;
281     new_psgr_lrec.lrec.stat.ir00mls = 0;        /* mileage credit   */
282     (void) ctime(&ltime);
283     tp = gmtime(&ltime);                        /* get date info    */
284     new_psgr_lrec.lrec.stat.ir00exy = tp->tm_year + 1; /* exp. year */
285     new_psgr_lrec.lrec.stat.ir00exm = tp->tm_mon;     /* exp. month */
286
287     add_psgr(psgr_file,&new_psgr_lrec);
288
289     if (DF_ER(psgr_file))
290       new_error();
291
292     (void) sprintf(msg_text,"%s%s",done_msg,next_num);
293     op_msg("SAM006I",msg_text);
294   }
295   return;
296 }
297
```

Figure 15. sam0.c–File Maintenance Program (Part 6 of 11)

```
298 /*--------------------------------------------------------------------*
299  *      Change Passenger Information                                  *
300  *--------------------------------------------------------------------*/
301 static void action_change(void)
302 {
303   /* A slash must follow the action code */
304   /* and the 10-character member number. */
305   if (in_msg[11]!= '/')
306     op_msg("SAM004E","INCORRECT FORMAT FOR CHANGE COMMAND");
307   else
308
309   /* Input format is OK, determine what is to change. */
310   {
311     switch (in_msg[12])
312     {
313       /* Change Passenger Address. */
314       case 'A':
315         change_address();
316         break;
317
318       /* Change Meal, Seat, or Payment Preference Information. */
319       case 'M':
320       case 'S':
321       case 'P':
322         change_info(in_msg[12]);
323         break;
324
325       /* Invalid Input. */
326       default:
327         op_msg("SAM003E","UNRECOGNIZED CHANGE CODE");
328         break;
329     }
330   }
331   return;
332 }
333
```

Figure 15. sam0.c–File Maintenance Program (Part 7 of 11)

```
/*--------------------------------------------------------------------*
 *      Change Passenger Address                                      *
 *--------------------------------------------------------------------*/
static void change_address(void)
{
  /* local variables */
  struct ir00df *psgr_lrec, new_psgr_lrec;
  const unsigned addr_size = strlen(&in_msg[13]); /* size of address */

  /* Verify the length of the address from the input message. */
  if (addr_size > MAX_ADDR_SIZE)
    op_msg("SAM007E","ADDRESS TOO LONG"); /* send message to operator */
  else

  /* Message length is OK, so continue. */
  {
    /* Read any existing passenger address record. */
    psgr_lrec = read_psgr_lrec(psgr_file,&in_msg[1],_IR00K90);

    if (DF_ER(psgr_file))               /* if error from read */
      subfile_error();                  /* doesn't return     */

    /* Build the new Passenger Address LREC. */
    new_psgr_lrec = *psgr_lrec;         /* copy old LREC */
    new_psgr_lrec.ir00siz = (short) (addr_size +
                                     sizeof(new_psgr_lrec.ir00siz) +
                                     sizeof(new_psgr_lrec.ir00key));
    (void) strncpy(new_psgr_lrec.lrec.addr.ir00adr,
                   &in_msg[13],
                   addr_size);

    /* Add or replace the Passenger Address LREC. */
    if (DF_NR(psgr_file))
    {
      add_psgr(psgr_file,&new_psgr_lrec);
    }
    else
    {
      replace_psgr(psgr_file,&new_psgr_lrec);
    }

    if (DF_ER(psgr_file))       /* if error from add or replace */
      subfile_error();          /* doesn't return               */
  }
  return;
}

```

Figure 15. sam0.c–File Maintenance Program (Part 8 of 11)

```
/*-------------------------------------------------------------------*
 *      Change Miscellaneous Passenger Information (meal, seat, pay). *
 *-------------------------------------------------------------------*/
static void change_info(char info_code)
{
  /* local variables */
  struct ir00df *psgr_lrec, new_psgr_lrec;
  unsigned data_size;                   /* expected size of data */
  const unsigned actual_data_size = strlen(&in_msg[13]);
  char *data_ptr;                       /* location of data      */

  /* Set up for specific information. */
  switch (info_code)
  {
    case 'M':
      data_size = MEAL_SIZE;
      data_ptr = new_psgr_lrec.lrec.info.ir00mpr;
      break;

    case 'S':
      data_size = SEAT_SIZE;
      data_ptr = new_psgr_lrec.lrec.info.ir00spr;
      break;

    case 'P':
      data_size = PMNT_SIZE;
      data_ptr = new_psgr_lrec.lrec.info.ir00pay;
      break;

    default:
      break;
  }

  /* Verify the length of the information from the input message. */
  if (actual_data_size != data_size)
    op_msg("SAM008E","INFORMATION INCORRECT LENGTH");
  else

```

Figure 15. sam0.c–File Maintenance Program (Part 9 of 11)

```
419   /* Message length is OK, so continue. */
420   {
421     /* Read any existing passenger information record. */
422     psgr_lrec = read_psgr_lrec(psgr_file,&in_msg<1>,_IR00KA0);
423
424     if (DF_ER(psgr_file))               /* if error from read */
425       subfile_error();                  /* doesn't return     */
426
427     /* If a record exists, copy it to the record build */
428     /* area, otherwise build a new, empty record.      */
429     if (!DF_NR(psgr_file))
430       new_psgr_lrec = *psgr_lrec;
431     else
432     {
433       new_psgr_lrec.ir00key = _IR00KA0;
434       new_psgr_lrec.ir00siz = member_size(struct ir00df,lrec.info)
435                             + member_size(struct ir00df,ir00siz)
436                             + member_size(struct ir00df,ir00key);
437       (void) memset(new_psgr_lrec.lrec.info.ir00mpr,' ',MEAL_SIZE);
438       (void) memset(new_psgr_lrec.lrec.info.ir00spr,' ',SEAT_SIZE);
439       (void) memset(new_psgr_lrec.lrec.info.ir00pay,' ',PMNT_SIZE);
440     }
441
442     /* Move the new data into the record build area. */
443     (void) memcpy(data_ptr,&in_msg<13>,data_size);
444
445     /* Add or replace the Passenger Information LREC. */
446     if (DF_NR(psgr_file))
447     {
448       add_psgr(psgr_file,&new_psgr_lrec);
449     }
450     else
451     {
452       replace_psgr(psgr_file,&new_psgr_lrec);
453     }
454
455     if (DF_ER(psgr_file))       /* if error from add or replace */
456       subfile_error();          /* doesn't return               */
457
458     op_msg("SAM009I","DETAILS MODIFIED");
459   }
460   return;
461 }
462
```

Figure 15. sam0.c–File Maintenance Program (Part 10 of 11)

```
463  /*--------------------------------------------------------------------*
464   *      Increment Consecutive Member Number.                          *
465   *--------------------------------------------------------------------*/
466  static void update_next_available_number(struct ir00df
*del_nbr_rcd)
467  {
468    /* local variables */
469    long bin_num;                 /* binary version of number */
470    char str_num<11>;             /* string version of number */
471
472    /* Verify that a number is available. */
473    if (strncmp(del_nbr_rcd->lrec.cnum.ir00nuc,"9999999999",10) == 0)
474      data_base_full();           /* no number available */
475
476    /* Convert number to binary and increment it. */
477    (void) strncpy(str_num,del_nbr_rcd->lrec.cnum.ir00nuc,10);
478    str_num<10>= 0x00;
479    bin_num = strtol(str_num,NULL,10) + 1;
480
481    /* Convert it back to a string and store it back. */
482    (void) sprintf(str_num,"%.10ld",bin_num);
483    (void) strncpy(del_nbr_rcd->lrec.cnum.ir00nuc,str_num,10);
484
485    return;
486  }
487
488  /*--------------------------------------------------------------------*
489   *      Error Handling                                                *
490   *--------------------------------------------------------------------*/
491  static void subfile_error(void)
492  {
493    op_msg("SAM010E","ERROR IN SUBFILE");
494    exit(12);
495  }
496
497  static void new_error(void)
498  {
499    op_msg("SAM011E","ERROR IN NEW SUBFILE");
500    exit(12);
501  }
502
503  static void data_base_full(void)
504  {
505    op_msg("SAM012E","NO MEMBER NUMBERS AVAILABLE");
506    exit(12);
507  }
508
```

Figure 15. sam0.c–File Maintenance Program (Part 11 of 11)

Departure Control Interface Program

```
 1  /*-------------------------------------------------------------------*
 2   *       DEPARTURE CONTROL INTERFACE                                 *
 3   *          Updates the mileage flown and adjusts the expiration     *
 4   *          date of the membership based on an extra month's         *
 5   *          membership for each 1000 miles flown.  No errors are     *
 6   *          issued from this program.  Indicators are returned       *
 7   *          as shown below.                                          *
 8   *-------------------------------------------------------------------*
 9   *       Input Parameters:                                           *
10   *          char mem_num[10]    - membership number                  *
11   *          short miles         - miles to be credited               *
12   *                                                                   *
13   *       Return Values (int):                                        *
14   *         NO_SUBFILE - no subfile for specified member number       *
15   *         NO_MILEAGE - no mileage lrec in member's subfile          *
16   *         PSGR_ERROR - serious error on read                        *
17   *         0          - no error                                     *
18   *-------------------------------------------------------------------*/
19  #include <cdf.h>
20  #include <psgr.h>
22  #define NO_SUBFILE      4
23  #define NO_MILEAGE      8
24  #define PSGR_ERROR     16
25
26  #define read_psgr_lrec(file,alg,pky) \
27                (df_nbrkeys(&psgr_keys,1), \
28                 psgr_pky = pky, \
29                 df_setkey(&psgr_keys,1, \
30                           offsetof(struct ir00df,ir00key), \
31                           1,DF_EQ,&psgr_pky,0,DF_UPORG), \
32                 read_psgr(file,alg))
33
34  static dft_kyl psgr_keys;       /* passenger file key area          */
35  static dft_pky psgr_pky;        /* passenger file primary key       */
36
37  static dft_fil *psgr_file;      /* passenger file pointer           */
38
```

Figure 16. sam1.c–Departure Control Interface (Part 1 of 2)

```
39 /*--------------------------------------------------------------------*
40  *      Update the Passenger File Mileage and Expiration Information. *
41  *--------------------------------------------------------------------*/
42 int sam1(char *mem_num, short miles)
43 {
44   /* local variables */
45   static struct ir00df *psgr_lrec;      /* passenger record */
46   int rc = 0;                           /* return code      */
47
48   /* Passenger subfile must be opened with the HOLD option. */
49   psgr_file = open_psgr(HOLD);
50
51   /* Search for this member's mileage lrec.                 */
52   psgr_lrec = read_psgr_lrec(psgr_file,mem_num,_IR00KB0);
53
54   if (DF_ERX(psgr_file))                /* if error on read */
55     rc = PSGR_ERROR;
56   if (DF_TEST(psgr_file,DFC_ALG))  /* if subfile doesn't exist  */
57     rc = NO_SUBFILE;
58   else if (DF_NR(psgr_file))       /* if mileage LREC not found */
59     rc = NO_MILEAGE;
60   else
61   {
62     /* Update the membership expiration date based on the mileage. */
63     psgr_lrec->lrec.stat.ir00mls +=miles;      /* update miles     */
64                                     /* add 1 month per 1000 miles */
65     psgr_lrec->lrec.stat.ir00exm +=psgr_lrec->lrec.stat.ir00mls / 1000;
66     psgr_lrec->lrec.stat.ir00mls %=1000;       /* adjust mileage   */
67                                     /* add 1 year per 12 months   */
68     psgr_lrec->lrec.stat.ir00exy +=psgr_lrec->lrec.stat.ir00exm / 12;
69     psgr_lrec->lrec.stat.ir00exm %=12;         /* adjust months    */
70
71     modify_psgr(psgr_file);
72   }
73
74   /* Close the member file and return to */
75   /* the caller with the return code.    */
76   close_psgr(psgr_file,0);
77
78   return (rc);
79 }
```

Figure 16. sam1.c–Departure Control Interface (Part 2 of 2)

Monthly Maintenance Program

```
 1  /*--------------------------------------------------------------------*
 2   *       MONTHLY MAINTENANCE PROGRAM                                  *
 3   *          Deletes members records that have expired and adds        *
 4   *          the membership number to a subfile so it can be           *
 5   *          reissued.                                                 *
 6   *--------------------------------------------------------------------*/
 7  #include <stdlib.h>
 8  #include <time.h>
 9  #include <cdf.h>
10  #include <psgr.h>
11
12  #define op_msg(prefix,text) \
13   ((void) time(&ltime), \
14    printf("%s %s %s\n",prefix,ctime(&ltime),text))
15
16  #define close_all()     dfcls(NULL,DFCLS_ALL|DFCLS_ABORT)
17
18  #define read_psgr_lrec(file,pky,area) \
19                 (df_nbrkeys(&psgr_keys,1), \
20                  psgr_pky = pky, \
21                  df_setkey(&psgr_keys,1, \
22                            offsetof(struct ir00df,ir00key), \
23                            1,DF_EQ,&psgr_pky,0,DF_UPORG), \
24                  read_psgr_full(file,area))
25
26  static time_t ltime;            /* used for time conversion          */
27
28  static dft_kyl psgr_keys;       /* passenger file key area           */
29  static dft_pky psgr_pky;        /* passenger file primary key        */
30
31  static dft_fil *psgr_file;      /* passenger file pointer            */
32  static dft_fil *del_nbr_file;   /* 'deleted numbers' subfile         */
33
34  static void update_free_slot(char *);
35  static void add_error(void);
36  static void read_error(void);
37
```

Figure 17. sam2.c–Monthly Maintenance Program (Part 1 of 3)

```
/*--------------------------------------------------------------------*
 *      Perform Monthly Maintenance on the Member File.               *
 *--------------------------------------------------------------------*/
void sam2(void)
{
  /* local variables */
  struct ir00df *psgr_lrec;             /* passenger record           */
  struct tm *tp;                        /* current time structure     */
  char mem_num.<MEM_NUM_SIZE>;          /* member number              */

  /* Restrict operation of this program to the 1st day of the month. */
  (void) ctime(&ltime);
  tp = gmtime(&ltime);                  /* get date info              */

  if (tp->tm_mday != 1)                 /* if not first of month... */
  {
    op_msg("SAM201E","NOT FIRST OF MONTH - UNABLE TO PROCESS");
  }
  else
  {
    /* Open the member file and the 'deleted numbers' subfile. */
    psgr_file = open_psgr(HOLD);
    del_nbr_file = open_del_nbr(HOLD);

    /* Read the member file until the end. */
    while (1)
    {
      /* Read each status lrec for the expiration date.   */
      /* The 'area' parameter provides the member number. */
      psgr_lrec = read_psgr_lrec(psgr_file,_IR00KB0,mem_num);

      /* Check for error or end of file. */
      if (DF_ERX(psgr_file))
        read_error();

      if (DF_EF(psgr_file))
        break;

      /* If the membership has expired, delete the subfile and add */
      /* the deleted number to the 'deleted numbers' subfile.      */
      if ((psgr_lrec->lrec.stat.ir00exm < tp->tm_mon &&
           psgr_lrec->lrec.stat.ir00exy <= tp->tm_year) ||
           psgr_lrec->lrec.stat.ir00exy < tp->tm_year)
      {
        delete_psgr(psgr_file,mem_num);
        update_free_slot(mem_num);
      }
    }
```

Figure 17. sam2.c–Monthly Maintenance Program (Part 2 of 3)

```
 87 |  |  op_msg("SAM202I","PROCESSING COMPLETED");
 88 |  |
 89 |  |  /* Close both files. */
 90 |  |  close_psgr(psgr_file,0);
 91 |  |  close_del_nbr(del_nbr_file);
 92 |  └}
 93 |
 94 |  exit(0);
 95 └}
 96
 97  static void update_free_slot(char*mem_num)
 98 ┌{
 99 |  static struct ir00df free_slot ={ _IR00LC0, _IR00KC0 };
100 |
101 |  /* Copy the member number into the free slot LREC. */
102 |  (void) memcpy(free_slot.lrec.rnum.ir00num,
103 |               mem_num,
104 |               member_size(struct ir00rnum,ir00num));
105 |
106 |  /* Add the free slot LREC to the deleted numbers subfile. */
107 |  add_del_nbr(del_nbr_file,&free_slot);
108 |
109 |  if (DF_ER(del_nbr_file))
110 |    add_error();
111 |
112 |  return;
113 └}
114
115  /*-------------------------------------------------------------------*
116   *      Error Handling                                               *
117   *-------------------------------------------------------------------*/
118  static void add_error(void)
119 ┌{
120 |  op_msg("SAM203E","ADD ERROR ON FILE IR00DF - JOB TERMINATED");
121 |  close_all();
122 |  exit(12);
123 └}
124
125  static void read_error(void)
126 ┌{
127 |  op_msg("SAM204E","READ ERROR ON FILE IR00DF - JOB TERMINATED");
128 |  close_all();
129 |  exit(12);
130 └}
131
```

Figure 17. sam2.c–Monthly Maintenance Program (Part 3 of 3)

Part 2. C Language Functions

TPFDF General-Use C Language Functions: Reference 71
dfadd–Add a Logical Record to a Subfile 73
dfadr–Provide the File Address of a Prime Block 80
dfckp–Checkpoint a Subfile 83
dfclr–Allow ECB Exit with Open Files 85
dfcls–Close a Subfile. 86
dfcpy–Copy a Subfile 92
dfcre–Create a Subfile 95
dfdel–Delete One or More Logical Records 97
dfdix–Delete Index References to a Subfile 105
dfdsp–Display Logical Records from a Subfile 107
dffrl–Ensure an ECB Data Level Is Free 111
dfidx–Create an Index Reference 112
dfifb–Check a SW00SR Slot 114
dfkey–Activate a Key List. 115
dfmod–Perform or Indicate Logical Record Modifications 117
dfmrg–Merge Logical Records from Two Subfiles 121
df_nbrkeys–Setting Up the Number of Keys 124
dfopn–Open a Subfile 125
dfopt–Set Optional Information. 130
dfred–Read a Logical Record 134
dfrep–Replace a Logical Record with Another Logical Record 143
dfret–Retain a Logical Record Position. 145
dfrst–Restore a Subfile 147
df_setkey–Setting Up a Key in a Key List. 150
dfspa–Create Work Space 156
dfsrt–Sort a Subfile 157
dftld–Write a Subfile from Main Storage to DASD. 160
dftlg–Write a File or Subfile to Tape 163
dftrd–Read a Subfile from an Input Tape to Main Storage 166
dfuky–Generate a Unique Key for Use in Logical Records 167
member_size–Calculating the Size of a Structure Member 168

TPFDF Restricted C Language Functions: Reference 169
dftab–Access Database Definition Tables 170

© Copyright IBM Corp. 1997, 2001

TPFDF General-Use C Language Functions: Reference

Note to ALCS Customers

You must enable TPFDF C language support before using the TPFDF C language functions. See *TPFDF Installation and Customization* for more information about enabling TPFDF C language support in an ALCS environment.

The TPFDF C language functions are set up in groups. All the functions in any one group perform a similar service. For example, each function in the `dfrst` function group restores a subfile that you specify. The specific function in a group that you use depends on what parameters you want to specify. For example, when you are restoring a subfile:

- Use the `dfrst` function if you want to restore the current subfile.
- Use the `dfrst_acc` function if you want to use an ordinal number, file address, or pointer to an algorithm argument to access the subfile.
- Use the `dfrst_seq` function if you want to perform sequence number checking.
- Use the `dfrst_acc_seq` function if you want to use an ordinal number, file address, or pointer to an algorithm argument to access the subfile, and perform sequence number checking.

This chapter contains an alphabetic listing of the TPFDF C language function groups that you can use in application programs. The description of each function group includes the following information:

Format: Provides the function prototype and a description of each parameter and variable.

Entry Requirements: Lists any special conditions that must be true when you use the function.

Normal Return: Lists what is returned when the function has completed processing successfully.

Error Return: Lists what is returned when the function cannot complete processing successfully.

Programming Considerations: Lists any additional considerations for using the function, including any restrictions or limitations.

Examples: Provides one or more examples that show you how to code the function.

Related Functions: Lists where to find information about related functions.

© Copyright IBM Corp. 1997, 2001

Include the Correct Header Files

Code the following statement in all your TPFDF C language application programs:

```
#include <cdf.h>
```

This statement causes all the necessary header files to be included in the C application program in the correct order. The following is a list of these header files:

- c$cdfeq.h
- c$cdfapi.h
- c$cdfmac.h
- c$cdflnk.h
- c$cdferr.h
- c$sw00sr.h
- c$sw01sr.h
- c$sw02sr.h

For information about the header files needed for TPF C support or ALCS C support, see *TPF Application Programming* or the *ALCS Application Programming Guide*.

dfadd–Add a Logical Record to a Subfile

Use this group of functions to:

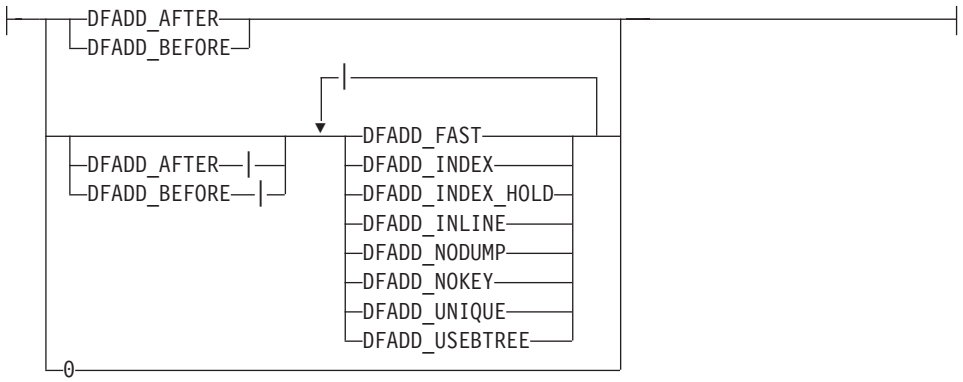
- Add a fixed- or variable-length logical record (LREC) to a subfile
- Add an empty LREC to a subfile
- Add a subLREC to the current extended LREC
- Add an extended LREC to a subfile
- Add an extended LREC and a subLREC to a subfile.

Format

```
dft_rec *dfadd(dft_fil *file, dft_opt rec_type, dft_opt options,
        dft_rec *rec);

dft_rec *dfadd_acc(dft_fil *file, dft_opt rec_type, dft_opt access,
        dft_opt options, dft_rec *rec, dft_xxx acc);

dft_rec *dfadd_nbr(dft_fil *file, dft_opt rec_type, dft_opt options,
        dft_rec *rec, dft_nbr nbr);

dft_rec *dfadd_pky(dft_fil *file, dft_opt rec_type, dft_opt options,
        dft_rec *rec, dft_pky pky);

dft_rec *dfadd_acc_nbr(dft_fil *file, dft_opt rec_type, dft_opt access,
        dft_opt options, dft_rec *rec, dft_xxx acc, dft_nbr nbr);

dft_rec *dfadd_acc_pky(dft_fil *file, dft_opt rec_type, dft_opt access,
        dft_opt options, dft_rec *rec, dft_xxx acc, dft_pky pky);

dft_rec *dfadd_nbr_pky(dft_fil *file, dft_opt rec_type, dft_opt options,
        dft_rec *rec, dft_nbr nbr, dft_pky pky);

dft_rec *dfadd_acc_nbr_pky(dft_fil *file, dft_opt rec_type,
        dft_opt access, dft_opt options, dft_rec *rec, dft_xxx acc,
        dft_nbr nbr, dft_pky pky);

dft_rec *dfadd_sub(dft_fil *file, dft_opt options,
        dft_rec *sub);

dft_rec *dfadd_usr_pky(dft_fil *file, dft_opt options,
        dft_rec usr, dft_pky pky);

dft_rec *dfadd_usr_acc_pky(dft_fil *file, dft_opt access,
        dft_opt options, dft_rec usr, dft_xxx acc,
        dft_pky pky);

dft_rec *dfadd_usr_nbr_pky(dft_fil *file, dft_opt options,
        dft_rec usr, dft_nbr nbr, dft_pky pky);

dft_rec *dfadd_usr_acc_nbr_pky(dft_fil *file, dft_opt access,
        dft_opt options, dft_rec usr, dft_xxx acc,
        dft_nbr nbr, dft_pky pky);

dft_rec *dfadd_usr_sub_pky(dft_fil *file, dft_opt options,
        dft_rec *usr, dft_rec *sub, dft_pky pky);

dft_rec *dfadd_usr_acc_sub_pky(dft_fil *file, dft_opt access,
        dft_opt options, dft_rec *usr, dft_xxx acc,
        dft_rec *sub, dft_pky pky);

dft_rec *dfadd_usr_sub_nbr_pky(dft_fil *file, dft_opt options,
        dft_rec *usr, dft_rec *sub, dft_nbr nbr,
        dft_pky pky);

dft_rec *dfadd_usr_acc_sub_nbr_pky(dft_fil *file, dft_opt access,
        dft_opt options, dft_rec *usr, dft_xxx acc,
        dft_rec *sub, dft_nbr nbr, dft_pky pky);
```

Access Parameter Values:

Options Parameter Values:



Rec_Type Parameter Values:

acc
is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
is the method you want to use to access the subfile. Use one of the following values:

DFADD_ALG
specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFADD_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFADD_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

nbr
is a variable containing the LREC sequence number in the subfile. (You can add an LREC or an extended LREC by providing a specific sequence number.)

Notes:

1. Do not use this parameter with files for which default keys are defined.
2. If you use the #TPFDB0D algorithm, you must specify this parameter.
3. LRECs are numbered in increasing order from the start of the subfile (the first LREC in the prime block has sequence number 1).
4. If you specify this parameter with active keys, only those LRECs that match the key conditions are included in the sequence numbering; LRECs that do not match are ignored.
5. When you specify this parameter, the LREC is added immediately ***after*** the specified LREC.
6. If you specify this parameter for an LREC number that does not exist, the LREC is not added.

options
are the processing options for this function. Use the following values:

DFADD_AFTER
adds the new LREC immediately after the current LREC.

DFADD_BEFORE
adds the new LREC immediately before the current LREC.

DFADD_FAST
used for migration purposes only; use the DFADD_INLINE or DFADD_NOKEY value instead. If you specify this parameter, the DFADD_NOKEY parameter is implemented; that is, any currently active keys are deactivated.

DFADD_INDEX
adds an LREC to a detail subfile or intermediate index subfile where the index structure does not yet exist. If you specify this value, the algorithm defined for the new subfile must be #TPFDBFF.

When you specify this parameter, the subfile is created and indexed by adding an index LREC in the index file referencing the subfile.

DFADD_INDEX_HOLD
potentially holds any index files that reference the subfiles you are accessing and prevents two or more application programs from modifying

the index files at the same time. Holding occurs if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the index file will not occur until the index file is no longer held. If more than one application can update the same index file, you must specify this value to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT marco, or the OP2= parameter in the DBDEF macro, affect hold processing.

DFADD_INLINE
provides a faster method for adding LRECs to a subfile. You cannot use this value with key parameters or with extended LRECs. Any keys that are active from previous TPFDF functions are deactivated.

DFADD_NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this function:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the DFADD_NODUMP value is not recommended because it can prevent system errors from being issued that indicate a critical problem.

DFADD_NOKEY
deactivates any currently active keys.

DFADD_UNIQUE
specifies that the TPFDF product should not add the LREC to the subfile if an LREC exists with the same key fields. The key fields are those specified by the currently active keys.

Notes:

1. If UNIQUE=YES was specified in the DBDEF macro for a file, all `dfadd` function calls for that file default to DFADD_UNIQUE.
2. If you specify this option for a file that is not organized (that is, defined as NOORG), you must do one of the following:
   - Define default keys for the file being updated
   - Ensure there are active keys from a previous `dfkey` function.

   If you do not have default keys defined or have active keys when adding a unique LREC to a file that is not organized, the TPFDF product issues a system error. Symbol &DB013E in the DBLCL macro controls whether the TPFDF product returns control to the application program or exits the entry control block (ECB) after issuing the error. If you set &DB013E to 0 (which is the default setting) the ECB exits. If

you set &DB013E to 1, control is returned to the application program. See *TPFDF Installation and Customization* for more information about the DBLCL macro.

DFADD_USEBTREE
specifies that the B+Tree index is used when adding an LREC to a subfile. You can use this value only on a B+Tree file. Otherwise, this value is ignored by the TPFDF product.

0 specifies that you do not want to use any processing options.

pky
is the primary key of the LREC that you are adding. You must specify this parameter when using the DFADD_NULLREC value of the *rec_type* parameter.

When adding an extended LREC, this primary key is placed in the *zzzz*KEY field in the control area of the extended LREC.

rec
is a pointer to the new LREC that you are adding.

rec_type
is the type of LREC that you are adding. Use one of the following:

DFADD_NEWLREC
adds a new fixed-length or variable-length LREC.

DFADD_NULLREC
adds an empty LREC that can be used as a work area.

Do not use this value for any file that is UP or DOWN organized because it can destroy the organization of the file.

sub
is a pointer to the subLREC that you are adding.

usr
is a pointer to the userLREC that you are adding.

Entry Requirements

- Before using the DFADD_AFTER or DFADD_BEFORE value, you must first establish a *current LREC* (for example, using a `dfred` function). You can then specify whether you want to add the new LREC before or after this current LREC by using the DFADD_BEFORE or DFADD_AFTER value for the *options* parameter on the `dfadd` function. You can add any number of LRECs at this point in a subfile without reestablishing the current LREC. The last LREC added becomes the current LREC.
- Before using the `dfadd_sub` function, you must first establish the extended LREC to which you want to add the subLREC as the current LREC.

Normal Return

One of the following:
- Pointer to the LREC that was added
- Pointer to the extended LREC that was added
- Pointer to the extended LREC to which a subLREC was added.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

- If there are default keys defined in the database definition (DBDEF) and you use the dfadd function with an LREC ID that has not been defined as a default key, a system error is issued.

Programming Considerations

- The type definitions (for example, dft_fil, dft_ref, and dft_kyl) are defined in the c$cdfapi.h header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as dft_*xxx*. See the description of the specific parameter for information about what type definition to use for that parameter.
- Do not use the following values for B+Tree files:
  - DFADD_AFTER
  - DFADD_BEFORE
  - DFADD_FAST
  - DFADD_NOKEY
  - DFADD_NULLREC.
- Do not use the DFADD_FAST value with extended LRECs; that is, do not use this value with the dfadd_sub function or any of the dfadd_usr_ functions.
- Do not use the following values with the dfadd_sub function:
  - DFADD_AFTER
  - DFADD_BEFORE
  - DFADD_UNIQUE.
- If keys are active for an add current file when you call a function from the dfadd group, the keys will not be used to determine the location of the record being added to the subfile. However, the keys will remain active for any subsequent functions. See *TPFDF Database Administration* for information about add current files.
- Any active keys are ignored when you use the #TPFDB0D algorithm.
- To add an LREC when no subfile is defined, the TPFDF product obtains a prime block from pool and inserts the LREC into it. It puts the address of this prime block in the SW00FAD field of the SW00SR slot and the record code check into the SW00WCC field.
- If adding an LREC to a subfile block causes the block to overflow, the TPFDF product gets a new block and chains it to the old one.

  Note: How LRECs are added to a subfile depends on the following factors:
  - For *add current files*, the LREC is added to the end of the subfile. See *TPFDF Database Administration* for more information about add current files.
  - For *pushdown chaining* files, the LREC is added as the last LREC in the prime block of the subfile. See *TPFDF Database Administration* for more information about pushdown chaining files.
  - For P-type files, a new block is added after the current block or at the end of the subfile.
  - If you specify the DFADD_AFTER or DFADD_BEFORE value, the LREC is added after or before the current LREC.
  - If you have active keys (which can include default keys coded on the DBDEF macro for the file), the LREC is added to the subfile at the specified location.
  - In all other cases, the LREC is added at the end of the subfile.

- You can use the DFADD_NEWLREC value for the *rec_type* parameter with a P-type file to specify the data contents of a new block.
- You can use the DFADD_NULLREC value for the *rec_type* parameter with a P-type file to add an empty block, chained to the current block. You can then add data to this block using the `dfmod` function.
- Use the `dfadd` function with the DFADD_NULLREC value for the *rec_type* parameter to create a work area for a T-type LREC in the underlying W-type file. Use the `dfdel` function to delete the T-type LREC from the underlying W-type file before exiting the application program.

Examples

- The following example adds LREC GR95SR to the current subfile. The new LREC is in `lrec`.

```
dft_fil *file_ptr;
struct gr95sr lrec;
   .
   .
   .
(void) dfadd(file_ptr, DFADD_NEWLREC, 0, &lrec);
```

- The following example adds LREC GR95SR to a subfile specified with an algorithm. The new LREC is in `lrec`. The algorithm argument is in `alg_string`.

```
dft_fil *file_ptr;
char alg_string[10] = "JONES";
struct gr95sr lrec;
   .
   .
   .
(void) dfadd_acc(file_ptr, DFADD_NEWLREC, DFADD_ALG, 0, &lrec, alg_string);
```

- The following example adds a subLREC to the current extended LREC.

```
dft_fil *file_ptr;
   .
   .
   .
dfadd_sub(file_ptr, 0, &sublrec);
```

- The following example adds an extended LREC to a subfile specified with an algorithm. The userLREC is in *userlrec* and the algorithm argument is in *alg_string*. The primary key of the extended LREC is hexadecimal 80.

```
dft_fil *file_ptr;
char alg_string[10]="JONES";
dft_pky pky;
pky=0x80;
   .
   .
   .
dfadd_usr_acc_pky(file_ptr, DFADD_ALG, 0, &userlrec, alg_string, pky);
```

- The following example shows how you can add new LRECs to a detail file by specifying an index key as the algorithm argument. To add passenger MCKAY to the passenger file, you can code:

```
dbptr=dfopn("GR45DF  ","S0",DFOPN_HOLD);
alg_string="MCKAY";
dfadd_acc(dbptr,DFADD_NEWLREC, DFADD_ALG, DFADD_INDEX, newlrec, alg_string);
```

Related Functions

- “dfdel–Delete One or More Logical Records” on page 97
- “dfmod–Perform or Indicate Logical Record Modifications” on page 117
- “dfopn–Open a Subfile” on page 125
- “dfred–Read a Logical Record” on page 134.

dfadr–Provide the File Address of a Prime Block

Use this group of functions to get the file address of a prime block in a fixed file.

You can also use a `dfadr` function to specify a range of ordinals to be used in subsequent fullfile processing.

Format

```
void dfadr_alg(dft_fil *file, df_opt options, dft_alg *alg);
void dfadr_ord(dft_fil *file, df_opt options, dft_ord ord);
void dfadr_beg(dft_fil *file, df_opt options, dft_alg *beg);
void dfadr_end(dft_fil *file, df_opt options, dft_alg *end);
void dfadr_beg_end(dft_fil *file, df_opt options, dft_alg *beg,
     dft_alg *end);
```

Options Parameter Values:

alg
is a pointer to an algorithm argument that identifies the subfile.

beg
is a pointer to an algorithm argument that is used to calculate the begin ordinal of the file. This ordinal is used as the start ordinal during fullfile processing.

end
is a pointer to an algorithm argument that the TPFDF product uses to calculate the end ordinal of the file. This ordinal is used as the end ordinal during fullfile processing.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use the following values:

DFADR_NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this function:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the DFADR_NODUMP value is not recommended because it can prevent system errors from being issued that indicate a critical problem.

DFADR_WRAPAROUND
reads LRECs from the start of the file to the end until it has read the whole file. Use this parameter value only when you intend to use fullfile processing.

For example, consider a file that contains five subfiles and the current subfile is number 3. If you specify DFADR_WRAPAROUND, and then call a `dfred` function with DFRED_FULLFILE, LRECs would be read from the subfiles in the order: 3, 4, 0, 1, 2.

0 specifies that you do not want to use any processing options.

ord
is the ordinal number of the subfile that you want to access.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

Entry Requirements

None.

Normal Return

The `dfadr` function does not change the current LREC even if you specify a different value for the *alg* or *ord* parameter with the `dfadr` function from that which was used to locate the LREC.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- You must specify a pointer to an algorithm argument for all the `dfadr` functions. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

  If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Examples

- The following example finds the file address of ordinal number 3 of a file.

```
dft_fil *file_ptr;
   .
   .
   .
dfadr(file_ptr, DFADR_ORD, 0, 3);
```

- The following example finds the file address of a subfile identified by the algorithm argument held in `member_number`.

```
dft_fil *file_ptr;
   .
   .
   .
dfadr(file_ptr, DFADR_ALG, 0, member_number);
```

Related Functions

- “dfdel–Delete One or More Logical Records” on page 97
- “dfdsp–Display Logical Records from a Subfile” on page 107
- “dfmod–Perform or Indicate Logical Record Modifications” on page 117
- “dfmrg–Merge Logical Records from Two Subfiles” on page 121
- “dfred–Read a Logical Record” on page 134
- “dfsrt–Sort a Subfile” on page 157
- “dftlg–Write a File or Subfile to Tape” on page 163.

dfckp–Checkpoint a Subfile

Use this function to checkpoint a subfile; that is, all blocks in main storage that have been changed are copied to DASD.

Format

```
dft_rec *dfckp(dft_fil *file, dft_opt options);
```

Options Parameter Values:

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use one of the following values:

DFCKP_DETAC
places the subfile in detac mode after checkpointing the subfile. When the subfile is in detac mode, all modified blocks are saved in main storage. Any changes that you make to the LRECs in that subfile are not written to DASD until the subfile is checkpointed or closed. You can discard modified LRECs (prevent them from being written to DASD) by using the DFCLS_ABORT value on the *options* parameter of the `dfcls` function.

Note: The TPF system and the ALCS environment issues a 000010 system error if an application program does not give up control in the time allotted by the application time-out counter. When processing in detac mode, a TPFDF application program can require more than the allotted time on a database with a large data structure. To prevent the 000010 system error, you can change the setting of the &TPFDBDV symbol in the DBLCL macro.

See *TPFDF Installation and Customization* for more information about the &TPFDBDV symbol and the DBLCL macro.

DFCKP_NODET
specifies that you do not want the subfile in detac mode after it has been checkpointed.

DFCKP_TM
specifies that commit scopes are used during checkpoint processing, regardless of what the database definition (DBDEF) macro has set as the default. This option is valuable when you want to file out many files during checkpoint processing (for example, detac mode, extensive B^+Tree indexing updates, and requests that result in packing).

DFCKP_NO_TM
specifies that commit scopes are not used during checkpoint processing, regardless of what the DBDEF macro has set as the default.

0 specifies that you do not want to use any processing options.

Entry Requirements

None.

Normal Return

The subfile remains open and the current logical record (LREC) remains the same.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- If you do specify the DFCKP_DETAC or the DFCKP_NODET value, the subfile remains in the mode currently being used.
- You can use the `dfckp` function to checkpoint a W-type file to a short-term pool file.
- For a B+Tree file, the `dfckp` function checkpoints the index blocks and the data blocks.
- Processing the `dfckp` function on a file that is opened in a commit scope is not visible until the file is committed. If the checkpointed file is rolled back, updates to the file are discarded. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

- The following example checkpoints the current subfile.

```
dft_fil *file_ptr;
   .
   .
   .
(void) dfckp(file_ptr, 0);
```

- The following example checkpoints a subfile and ensures the subfile is in detac mode after processing:

```
dft_fil *file_ptr;
   .
   .
   .
(void) dfckp(file_ptr, DFCKP_DETAC);
```

Related Functions

- “dfcls–Close a Subfile” on page 86
- “dfopn–Open a Subfile” on page 125.

dfclr–Allow ECB Exit with Open Files

Use this function to allow a program to exit without closing any open subfiles and without generating a dump.

Attention: We do not recommend using this function because it can leave subfiles in a partially updated condition. Instead, use the `dfcls` function with the DFCLS_ABORT value of the *options* parameter if you want to discard updates made while a subfile is in detac mode.

Format

```
void dfclr();
```

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

None.

Examples

The following example allows the program to exit with files still open.

```
dfclr();
```

Related Functions

“dfcls–Close a Subfile” on page 86.

dfcls–Close a Subfile

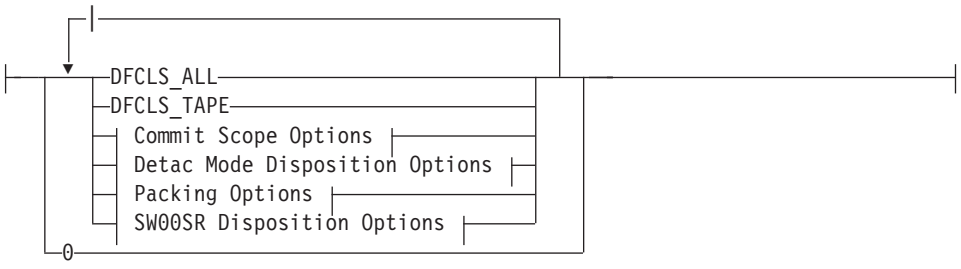
Use this group of functions to close one or more subfiles. You can also choose whether you want to write modified blocks that are in detac mode to DASD.

Format

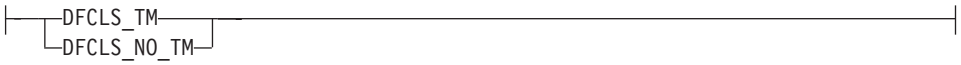
```
void dfcls(dft_fil *file, dft_opt options);
void dfcls_lst(dft_fil *file, dft_opt list_type, dft_opt options,
       dft_rfl *lst);
void dfcls_alg(dft_fil *file, dft_opt options, dft_alg *alg);
void dfcls_new(dft_fil *file, dft_opt options, dft_ref new);
```

List_Type Parameter Values:

Options Parameter Values:

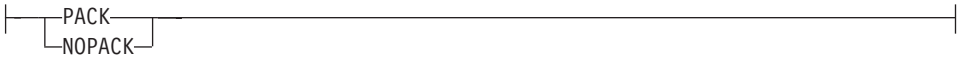


Commit Scope Options:



Detac Mode Disposition Options:

Packing Options:



SW00SR Disposition Options:

alg
is a pointer to an algorithm argument that identifies the subfile.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based

on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

If you specify the DFCLS_ALL value for the *options* parameter, specify NULL for the *file* parameter.

list_type
specifies which files to close. Use one of the following values:

DFCLS_INCLUDE
closes only the files listed in the structure pointed to by the *lst* parameter. If any of the listed files are not open, they are ignored.

DFCLS_EXCLUDE
closes all files except those listed in the structure pointed to by the *lst* parameter.

Do not specify DFCLS_INCLUDE or DFCLS_EXCLUDE if you use the DFCLS_ALL value for the *options* parameter.

lst is a pointer to a structure that contains a list of files.

new
is a pointer to a new reference name. Specify this parameter only if you specify the DFCLS_REUSE or DFCLS_NORELEASE values for the *options* parameter.

options
are the processing options for this function. Use the following values:

DFCLS_ABORT
causes all database updates since the file was opened, or since the last `dfckp` function, to be discarded. The updates are not written to DASD.

Note: The DFCLS_ABORT option is ignored if the subfile is not opened in detac mode because all updates have been written to DASD by previous TPFDF functions.

DFCLS_COMMIT
writes all blocks that have been modified since the last `dfopn` or `dfckp` function to DASD.

Note: The DFCLS_COMMIT option is ignored if the subfile is not opened in detac mode because all updates have been written to DASD by previous TPFDF functions.

DFCLS_RELFC
releases the subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

W-type files are automatically released unless they have been sorted, merged, or checkpointed. In these cases, you must specify the DFCLS_RELFC value to release W-type files.

DFCLS_RELEASE
releases the SW00SR slot when the subfile is closed.

DFCLS_NORELEASE
prevents the SW00SR slot from being released when the file is closed. Any key parameters you have defined are also retained. Specify this parameter if you intend to process the same subfile at a later time.

DFCLS_REUSE
retains the SW00SR slot of the file. Any key parameters you have defined are also retained. Specify this parameter if you intend to retrieve another subfile in the same file at a later time.

DFCLS_PACK
packs the subfile.

DFCLS_NOPACK
specifies that you do not want to pack the subfile, even if a block has fallen below the packing threshold defined by the PIN parameter on the DBDEF macro. See *TPFDF Database Administration* for more information about the packing threshold.

DFCLS_ALL
closes all open files.

Do not specify DFCLS_ALL if you use the DFCLS_INCLUDE or DFCLS_EXCLUDE value for the *list_type* parameter.

Note: When you use this parameter, all database interface blocks (DBIFBs), which contain SW00SR slots, are released. However, DBIFBs are ***not*** released if there are no files open or any of the following parameters have been specified:

- EXCLUDE
- INCLUDE
- NORELEASE
- REUSE.

DFCLS_TAPE
closes the tape or sequential data set. You must specify this value if you opened the subfile with a tape name (*tpn* parameter).

Note: Do not use this value with the DFCLS_TM value because the integrity of the commit scope could be compromised, and files that are on tape cannot be rolled back.

DFCLS_TM
specifies that commit scopes are used during close processing, regardless of what the database definition (DBDEF) macro has set as the default. This option is valuable when many files are to be filed out during close processing (for example, detac mode, extensive B^+Tree indexing updates, and requests that result in packing).

DFCLS_NO_TM
specifies that commit scopes are not used during close processing, regardless of what the DBDEF macro has set as the default.

0 specifies that you do not want to use any processing options.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Do not check the error indicators in the SW00RTN field of the SW00SR slot. When you close a subfile, the `dfcls` function releases the SW00SR slot for the file (unless you specify the DFCLS_NORELEASE or DFCLS_REUSE values), so the error indicators are not available for you to check. When you specify the DFCLS_NORELEASE or DFCLS_REUSE values the SW00SR slot is not released, but the error indicators are not valid.
- Deleting LRECs from a subfile leaves empty space in the blocks. If you do not specify the DFCLS_PACK or DFCLS_NOPACK value and the file is not a B$^+$Tree file, the subfile is packed if the amount of space occupied by LRECs in any block falls below the threshold defined in the database definition (DBDEF).

  B$^+$Tree files are not packed during close processing unless the DFCLS_PACK value is specified or there are no nodes in the B$^+$Tree index.
- It is not necessary to close T-type files. In addition, it is not necessary to close W-type file that are open in detac mode. See *TPFDF Database Administration* for more information about T-type and W-type files.
- The `dfcls` functions do an internal `dfdix` function when you specify a DFCLS_RELFC value in the *options* parameter.
- If you do not specify the DFCLS_INCLUDE value with the `dfcls_lst` function and an attempt is made to close a subfile that is not open, the TPFDF product issues a DB0115 system error.
- If you specify DFCLS_ALL with the DFCLS_TM value and more than one file is open, the files in a commit scope are processed individually.

Examples

- The following example closes all open subfiles.

```
dfcls(NULL, DFCLS_ALL);
```

- The following example closes a specific subfile and commits all changes. The alternative is to use DFCLS_ABORT.

```
dft_fil *file_ptr;
   .
   .
   .
dfcls(file_ptr, DFCLS_COMMIT);
```

- The following example closes a list of subfiles. If any of the listed subfiles are not open, the subfiles that are not open are ignored.

```
struct
{
  dft_rfl rl;
  char more_refs[5][8];  /* allows a total of 6 references */
```

```
  char ch;               /* room for zero byte            */
} close_list;

short int count = 3;

memcpy(&close_list,"  GR95SR  GR91SR  GR93SR  ",26);
memcpy(&close_list,&count,2);
dfcls_lst(NULL,DFCLS_INCLUDE,0,&close_list);
```

Related Functions

- “dfopn–Open a Subfile” on page 125
- “dfckp–Checkpoint a Subfile” on page 83.

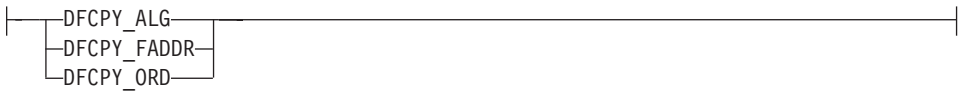
dfcpy–Copy a Subfile

Use this group of functions to copy a subfile. After processing the function, the TPFDF product closes the subfile and performs all subsequent actions on the copy.

Format

```
dft_hdr *dfcpy(dft_fil *file, dft_opt options);

dft_hdr *dfcpy_acc(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc);

dft_hdr *dfcpy_toa(dft_fil *file, dft_opt options, dft_fad toa);

dft_hdr *dfcpy_acc_toa(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc, dft_fad toa);

dft_hdr *dfcpy_acc_pth(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc, dft_pth pth);

dft_hdr *dfcpy_acc_toa_pth(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc, dft_fad toa, dft_pth pth);
```

Access Parameter Values:



Options Parameter Values:

acc
is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
is the method you want to use to access the subfile. Use one of the following values:

DFCPY_ALG
specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses

the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFCPY_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFCPY_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use the following values:

DFCPY_CREATE
creates a new subfile using pool blocks.

DFCPY_HELD
specifies that the entry control block (ECB) is already holding the file address specified by the *toa* parameter. You can use this value only if you specify a *toa* parameter.

0 specifies that you do not want to use any processing options.

pth
is the path number for a detail subfile using index support. The value is defined in the DBDEF macro and is a decimal number (0, 1, 2, and so on). The default path number is 0.

See *TPFDF Database Administration* for more information about path indexes.

toa
is an actual file address. The TPFDF product copies the subfile so that the prime block of the copy is at this address.

If used in a commit scope, the prime block specified by the *toa* parameter must be opened in the same commit scope as the subfile that is being copied. See “Commit Scopes” on page 8 for more information about commit scopes.

Entry Requirements

None.

Normal Return

A pointer to the main storage address of the header of the prime block of the copy of the subfile.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_`*`xxx`*. See the description of the specific parameter for information about what type definition to use for that parameter.
- This function sets up a 2-byte sequence number in the SW00SEQ field in the SW00SR slot. You can specify this sequence number to restore the same subfile with a subsequent `dfrst` function.
- When a B+Tree file is copied, the `dfcpy` function builds the B+Tree index for the new file.

Examples

The following example copies a subfile to pool blocks. (The TPFDF product sets a pointer to the header in block_ptr.)

```
dft_fil *file_ptr;
dft_hdr *block_ptr;
   .
   .
   .
block_ptr = dfcpy(file_ptr, 0);
```

Related Functions

“dfrst–Restore a Subfile” on page 147.

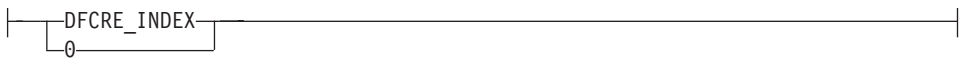
dfcre–Create a Subfile

Use this group of functions to create a new subfile, an empty pool subfile, or an empty indexed pool subfile with its corresponding index file. You can subsequently add logical records (LRECs) to the empty detail subfile as required.

Format

```
dft_hdr *dfcre(dft_fil *file, dft_opt options);
dft_hdr *dfcre_alg(dft_fil *file, dft_opt options, dft_alg *alg);
dft_hdr *dfcre_alg_pth_all(dft_fil *file, dft_opt options, dft_alg *alg);
dft_hdr *dfcre_alg_pth(dft_fil *file, dft_opt options, dft_alg *alg
        dft_pth pth);
```

Options Parameter Values:



alg
is a pointer to an algorithm argument that identifies the subfile.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use one of the following values:

DFCRE_INDEX
creates an indexed subfile and inserts an index LREC referencing this subfile into the related index file (or files) defined by the database administrator.

If you have supplied an algorithm argument when you opened the subfile (using a `dfopn` function), you can use DFCRE_INDEX with the `dfcre` function. If not, you must use the `dfcre_alg` function and supply an algorithm argument in the *alg* parameter.

If you specify this value, the algorithm defined for the new subfile must be #TPFDBFF.

0 specifies that you do not want to use any processing options.

pth
is the path number for a detail subfile using index support. The value is defined in the DBDEF macro and is a decimal number (0, 1, 2, and so on). The default path number is 0.

See *TPFDF Database Administration* for more information about path indexes.

Entry Requirements

None.

Normal Return

A pointer to the main storage address of the header of the prime block of the created subfile.

Error Return

See "Identifying Return Indicators and Errors" on page 13 for information about how to check the error indicators.

Programming Considerations

- Set the *options* parameter to zero if you do not require any processing options.
- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- A `dfadd` function creates a subfile, if one does not exist, before it adds the LREC. (If you are adding an LREC to an indexed detail file, you must use the DFADD_INDEX value of the *options* parameter or use a `dfidx` function to index the subfile.)

  Alternatively, you can call `dfcre` before you add an LREC to a subfile to create the subfile if it does not already exist.

Examples

The following example creates a new indexed subfile using the algorithm argument member_number.

```
dft_fil *file_ptr;
char member_number[10];
   :
   :
(void)dfcre_alg(file_ptr, 0, member_number);
```

Related Functions

- "dfadd–Add a Logical Record to a Subfile" on page 73
- "dfdix–Delete Index References to a Subfile" on page 105
- "dfidx–Create an Index Reference" on page 112.

dfdel–Delete One or More Logical Records

Use this group of functions to delete:

- One or more logical records (LRECs) from an open subfile
- The LRECs in one or more subfiles of a file
- A complete extended LREC
- All of the LRECs in a file
- Some or all of the subfiles referenced from the LRECs that you delete
- One or more subLRECs in an extended LREC.

Format

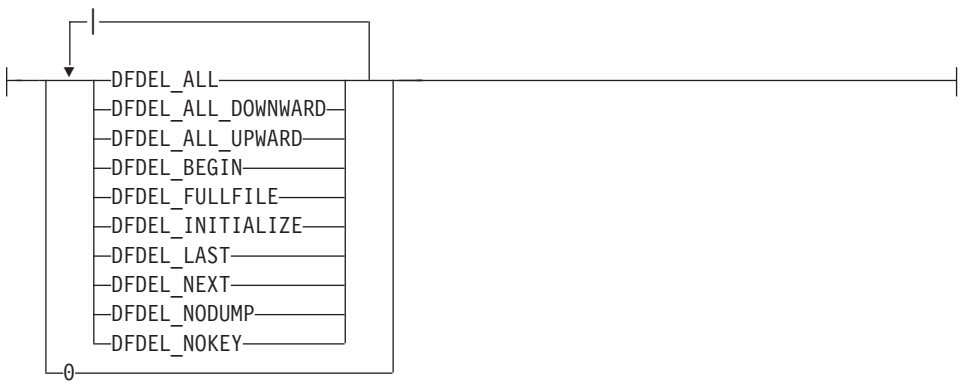
```
dft_rec *dfdel(dft_fil *file, dft_opt options);

dft_rec *dfdel_acc(dft_fil *file, dft_opt access, dft_opt options, dft_xxx acc);

dft_rec *dfdel_lst(dft_fil *file, dft_opt list_type, dft_opt options,
        dft_idl *lst);

dft_rec *dfdel_nbr(dft_fil *file, dft_opt nbr_type, dft_opt options,
        dft_xxx nbr);

dft_rec *dfdel_acc_lst(dft_fil *file, dft_opt list_type, dft_opt access,
        dft_opt options, dft_xxx acc, *dft_idl lst);

dft_rec *dfdel_acc_nbr(dft_fil *file, dft_opt nbr_type, dft_opt access,
        dft_opt options, dft_xxx acc, dft_xxx nbr);

dft_rec *dfdel_lst_nbr(dft_fil *file, dft_opt list_type, dft_opt nbr_type,
        dft_opt options, dft_idl *lst, dft_xxx nbr);

dft_rec *dfdel_acc_lst_nbr(dft_fil *file, dft_opt list_type,
        dft_opt nbr_type, dft_opt access, dft_opt options,
        dft_xxx acc, dft_idl *lst, dft_xxx nbr);

dft_rec *dfdel_sub(dft_fil *file, dft_opt options,
        dft_sno sub, dft_qty qty);
```

Access Parameter Values:

List_Type Parameter Values:

Nbr_Type Parameter Values:

Options Parameter Values:



acc
: is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
: is the method you want to use to access the subfile. Use one of the following values:

  DFDEL_ALG
  : specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFDEL_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFDEL_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

list_type
specifies which files to delete. Use one of the following values:

DFDEL_INCLUDE
deletes only the files listed in the structure pointed to by the *lst* parameter. This is the default value.

DFDEL_EXCLUDE
deletes all files except those listed in the structure pointed to by the *lst* parameter.

lst is a pointer to a list of up to 10 subfiles to be included or excluded in the delete action when the DFDEL_INCLUDE or DFDEL_EXCLUDE value of the *list_type* parameter is specified.

The *lst* parameter is provided so that you can delete, or not delete, subfiles that are referenced from the LREC that you are deleting.

To delete LRECs in all the referenced subfiles, specify zero as the number of file identifiers and provide one file identifier consisting of a string of 2 space characters. For example:

```
struct id_list ids = {0,"  "};
```

nbr
is an LREC number (type `dft_nbr`) or a pointer to a list (type `dft_rcl`) of LRECs to be deleted. The following is deleted based on the value of the *nbr_type* parameter:

- A single LREC specified by *nbr* if the *nbr_type* parameter is set to DFDEL_LRECNBR.
- All the LRECs in a list pointed to by *nbr* if the *nbr_type* parameter is set to DFDEL_LIST. Specify the LRECs in this list as character array. For example:

  ```
  char nbrs[10] = "2/3-6/LAST";
  ```

  The list contains one or more LREC sequence numbers separated by a slash (/). You can also specify a range of sequence numbers by separating the beginning and end of the range by a hyphen (-). You can use LAST to mean the last LREC of the subfile and ALL to mean all the remaining LRECs. You can also end the list with a nonnumeric character.

Notes:

1. The ranges must be in ascending order; if one is found out of order, that range and all subsequent ranges are ignored.

   For example, if there are 41 LRECs in a subfile, the following lists all have the same effect:

   ```
   20/31/32/33/37/38/39/40/41
   20/31/32/33/37-41
   20/31-33/37-LAST
   20/31-33/37/ALL
   ```

2. You ***cannot*** specify the number zero in the list of LREC numbers, even if you specify the ADJUST parameter with a value that would adjust the number zero to a valid LREC number.

Notes:

1. If you use the #TPFDB0D algorithm, you must specify a specific LREC number for the value of *nbr*.
2. LRECs are numbered in increasing order from the start of the subfile (the first LREC in the prime block has sequence number 1).
3. If you specify this parameter with active keys, only those LRECs that match the key conditions are included in the sequence numbering; LRECs that do not match are ignored.
4. In functions that do not include the *nbr_type* parameter, the value of *nbr* is a specific LREC number.

nbr_type
specifies the type of value that you are providing in the *nbr* parameter. Use one of the following values:

DFDEL_LIST
specifies that you are providing a pointer to a list of LREC numbers (type `dft_rcl`).

DFDEL_LRECNBR
specifies that you are providing the sequence number of a single LREC (type `dft_nbr`).

options
are the processing options for this function. Use the following values:

DFDEL_ALL
deletes every LREC in the open subfile that you have specified. If you currently have keys active, the function deletes only the LRECs that match the keys.

If you delete LRECs from a fixed file using the DFDEL_ALL value, the function writes the empty prime block to DASD after deleting the LRECs (the block header and optional trailer are not deleted). Any blocks previously chained to the prime block are released. It releases any blocks previously chained to the prime block.

If you delete all the LRECs from a pool subfile, the function releases both prime and overflow blocks. However, if you delete all the LRECs from a pool subfile and add an LREC before you close the subfile, the prime block is not released.

For B^+Tree files, if no keys are specified, DFDEL_ALL releases the index blocks as well as the data blocks.

If the subfile was opened in detac mode, you cannot recover the subfile using the `dfcls` function with DFCLS_ABORT.

If you combine this value with a *lst* parameter, the TPFDF product deletes only the LRECs in the *lst* parameter.

If you specify keys in addition to DFDEL_ALL, this function deletes all the LRECs in the open subfile that match these keys. If you also specify the DFDEL_FULLFILE value, the `dfdel` function deletes all the LRECs that match the keys in an entire file. In this case, the blocks are not released, whether they are fixed or pool, because some LRECs can remain in a file after deletion.

You can use the DFDEL_ALG value on the *access* parameter with the DFDEL_ALL value to delete LRECs from an indexed detail file. The DBDEL macro deletes the index entry for the subfile and releases the indexed subfile if it is in pool.

DFDEL_ALL_DOWNWARD
deletes all LRECs from (and including) the current LREC to the last LREC in the subfile. If you currently have keys active, the function deletes only the LRECs that match the keys.

DFDEL_ALL_UPWARD
deletes all LRECs from (but not including) the current LREC to the first LREC in the subfile. If you currently have keys active, the function deletes only the LRECs that match the keys.

DFDEL_BEGIN
specifies that you want to start at the beginning of the file when searching for LRECs to delete.

DFDEL_FULLFILE
deletes an LREC from every subfile of the file.

If you combine this value with the DFDEL_ALL value, the function deletes every LREC in every subfile of the file. You can delete LRECs in certain subfiles only by specifying the *beg* and *end* parameters with the `dfadr` function.

DFDEL_INITIALIZE
empties the entire subfile apart from the standard TPFDF header in the prime block. It releases any blocks previously chained to the prime block.

DFDEL_LAST
deletes the last LREC of the subfile. (If you have set keys, the function only deletes LRECs with matching keys.)

DFDEL_NEXT
deletes the next LREC in sequence from a file. (If you have set keys, the function only deletes LRECs with matching keys.)

DFDEL_NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this function:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the DFDEL_NODUMP value is not recommended because it can prevent system errors from being issued that indicate a critical problem.

DFDEL_NOKEY
deactivates any currently active keys.

Notes:

1. If the file is not open when you specify the DFDEL_NOKEY value, the `dfdel` function opens the file and deletes the *first* LREC.
2. If the file is open when you specify the DFDEL_NOKEY value, the `dfdel` function deletes the *current* LREC.

0 specifies that you do not want to use any processing options.

qty
is the number of subLRECs you want to delete.

sub
is the sequence number of the first subLREC in the extended LREC that you want to delete.

Entry Requirements

Before using the `dfdel_sub` function, you must use a `dfred` function to locate the extended LREC from which you want to delete the subLREC.

Normal Return

- One of the following:
  - Pointer to the next LREC in the subfile (after the last deleted LREC)
  - Pointer to the current extended LREC from which the subLREC or subLRECs were deleted.
- After you use the DFDEL_ALL_DOWNWARD or DFDEL_ALL_UPWARD value, the SW00REC is set to zero and the SW00RTN is set to X'40'.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.
- A pointer to protected main storage for all functions except the `dfdel_sub` function.
- For the `dfdel_sub` function, there is no error return when the subLRECs that you specify to be deleted do not exist. In all cases, the return value is as if the function was processed successfully.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_`*`xxx`*. See the description of the specific parameter for information about what type definition to use for that parameter.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent functions when

available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with functions that access or update records without using fullfile processing.

- If you call the `dfcls` function with an *options* value of DFCLS_RELFC, you do not have to call a `dfdel` function. In this case, the `dfcls` function does an internal delete.
- If you specify the DFDEL_FULLFILE value and the end-of-file indicator is set, you cannot call additional TPFDF functions until the file is closed. However, you can specify the DFCLS_REUSE value on the `dfcls` function. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- Any active keys are ignored when you use the #TPFDB0D algorithm.
- If you specify the DFDEL_ALL and DFDEL_NOKEY values for a B+Tree file, the `dfdel` function will pack the file.
- Before you use the DFDEL_ALL_DOWNWARD or DFDEL_ALL_UPWARD value, first establish a *current LREC* (for example, using the `dfred` function).

  Attention: Using the DFDEL_ALL_DOWNWARD or the DFDEL_ALL_UPWARD value can cause the values that were created previously for the `dfret` function to be corrupted.

Examples

- The following example deletes all LRECs in a subfile.

```
dft_fil *file_ptr;
   .
   .
   .
(void) dfdel(file_ptr, DFDEL_ALL);
```

- The following example deletes five subLRECs, starting at the first (subLREC number 0):

```
dft_fil *file_ptr;
   .
   .
   .
dfdel_sub(file_ptr, 0, 0, 5);
```

- You can delete a number of subLRECs without starting at the first. To delete *n* subLRECs from the *m*th subLREC onward, specify:

```
dft_fil *file_ptr;
   .
   .
   .
dfdel_sub(file_ptr, 0, m-1, n);
```

  (The first subLREC in an extended LREC is numbered zero so the *m*th subLREC is numbered *m-1*.)

- To delete all the subLRECs from the *m*th to the last in the LREC, specify:

```
dft_fil *file_ptr;
   .
   .
   .
dfdel_sub(file_ptr, 0, m-1, 99);
```

  (The previous example assumes there is a maximum of 99 subLRECs after the *m*th subLREC.)

- Suppose an extended LREC contains six subLRECs. You could delete the two middle subLRECs (numbers 2 and 3) by specifying:

```
dft_fil *file_ptr;
   .
   .
   .
dfdel_sub(file_ptr, 0, 2, 2);
```

Related Functions

- “dfcls–Close a Subfile” on page 86
- “dfmod–Perform or Indicate Logical Record Modifications” on page 117
- “dfred–Read a Logical Record” on page 134.

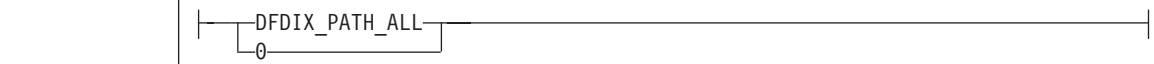
dfdix–Delete Index References to a Subfile

Use this group of functions to delete index references to the current subfile or a specific subfile.

Format

```
void dfdix_alg(dft_fil *file, dft_opt options, dft_alg *alg);
void dfdix_alg_pth(dft_fil *file, dft_opt options, dft_alg *alg,
     dft_alg pth);
```

Options Parameter Values:



alg
is a pointer to an algorithm argument that identifies the subfile.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use one of the following values:

DFDIX_PATH_ALL
deletes the index references for all paths.

0 specifies that you do not want to use any processing options.

pth
is the path number for a detail subfile using index support. The value is defined in the DBDEF macro and is a decimal number (0, 1, 2, and so on). The default path number is 0.

See *TPFDF Database Administration* for more information about path indexes.

Entry Requirements

Ensure that the relationship of the index file (or index files, if there are multilevel indexes) to the detail file has been defined with the DBDEF macro by your database administrator.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- The `dfcls` functions do an internal `dfdix` function when you specify a DFCLS_RELFC value in the *options* parameter.

Examples

The following example deletes an index reference to a subfile (path 1 only).

```
dft_fil *file_ptr;
char member_number[10];
   :
   :
dfdix_alg_pth(file_ptr, 0, member_number, 1);
```

Related Functions

- “dfcls–Close a Subfile” on page 86
- “dfcre–Create a Subfile” on page 95
- “dfidx–Create an Index Reference” on page 112.

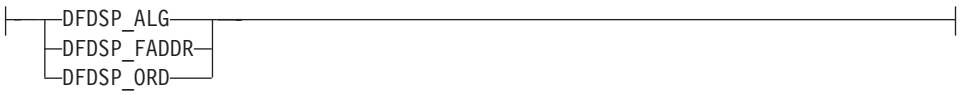
dfdsp–Display Logical Records from a Subfile

Use this group of functions to display the logical records (LRECs) from a subfile.

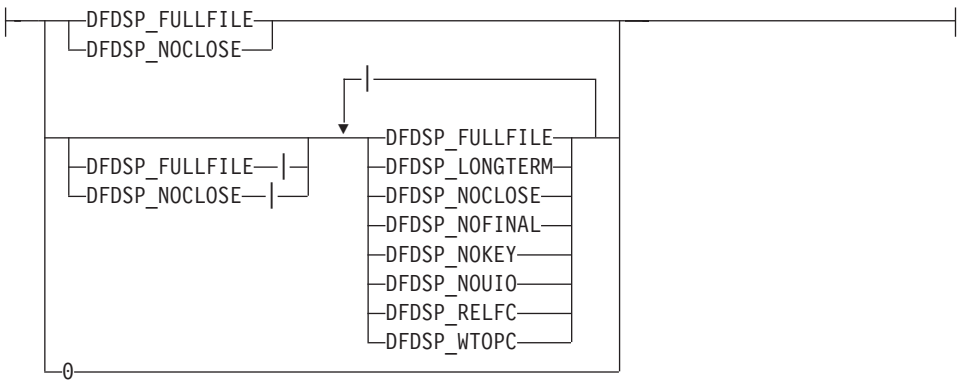
Format

```
void dfdsp(dft_fil *file, dft_opt options);

void dfdsp_acc(dft_fil *file, dft_opt access,
     dft_opt options, dft_xxx acc);

void dfdsp_str(dft_fil *file, dft_opt options, dft_str str);

void dfdsp_acc_str(dft_fil *file, dft_opt access,
     dft_opt options, dft_xxx acc, dft_str str);

void dfdsp_acc_opm(dft_fil *file, dft_opt access,
     dft_opt options, dft_xxx acc, dft_opm *opm);

void dfdsp_str_opm(dft_fil *file, dft_opt options,
     dft_str str, dft_opm *opm);

void dfdsp_acc_str_opm(dft_fil *file, dft_opt access, dft_opt options,
     dft_xxx acc, dft_str str, dft_opm *opm);
```

Access Parameter Values:



Options Parameter Values:



acc
: is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
: is the method you want to use to access the subfile. Use one of the following values:

DFDSP_ALG
: specifies that you are providing a pointer to an algorithm argument in the

acc parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFDSP_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFDSP_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

opm
is a pointer to the output message transmitter (OPMT) parameters (the 5-byte indicators for UIO).

options
are the processing options for this function. Use the following values:

DFDSP_FULLFILE
allows you to display LRECs from the whole file instead of from just one subfile. Do not use this value with W-type files or the DFDSP_NOCLOSE value.

DFDSP_LONGTERM
instructs the application program to use the MOSG internal program to build the output message (OMSG) display using long-term pool records. If you do not specify this value, short-term pool records are used for the display by the FMSG program.

DFDSP_NOCLOSE
specifies that you do not want to close the subfile displayed with the `dfdsp` function. This allows the application program to return to the open subfile once the function has completed processing. If you specify this value, ensure you specify that control is returned to the application program after the function completes its processing.

DFDSP_NOFINAL
indicates that this is only part of a message. The complete output message is displayed only when you call the `dfdsp` function without the DFDSP_NOFINAL value specified.

DFDSP_NOKEY
deactivates any currently active keys.

DFDSP_NOUIO
prevents the activation of the output edit CRT driver (UIO) and returns to the application program.

DFDSP_RELFC
releases the subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

W-type files are automatically released unless they have been sorted, merged, or checkpointed. In these cases, you must specify the DFDSP_RELFC value to release W-type files.

DFDSP_WTOPC
displays the LREC in WTOPC format. (By default, the function uses OMSG format.) This value indicates that the maximum length displayed for an LREC will be 255 bytes. The DFDSP_LONGTERM and DFDSP_NOUIO values and *opm* parameter are ignored.

0 specifies that you do not want to use any processing options.

str is a variable containing the number of bytes that you want to strip from the start of each LREC (or from the start of the user portion of an extended LREC).

Variable length LRECs contain a 2-byte size field at the front of the user data section. The `dfdsp` function automatically discards this field; do not include it in the number of bytes you specify with this parameter.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_`*xxx*. See the description of the specific parameter for information about what type definition to use for that parameter.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent functions when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with functions that access or update records without using fullfile processing.
- The TPFDF product does not normally return control to the application when you call `dfdsp` functions. The application exits unless you specify DFDSP_NOUIO or DFDSP_NOFINAL, or you specify not to exit using the *opm* parameter. However,

you must still test SW00RTN for errors because control is returned to the program if there are no LRECs to display, or if an error condition occurs.

Note: Although the TPFDF product preserves all data levels across TPFDF macro calls, the following exceptions exist when you use the `dfdsp` function:

- Data level 1 (D1) and data level 3 (D3) are ***not*** data level independent (DLI) if you do not specify the DFDSP_WTOPC value and DBLCL macro symbol &ACPDBAA is set to zero.
- Data level 2 (D2) is ***not*** DLI.

- You can limit the number of output lines displayed by the `dfdsp` function by using the #DF_MAX_DSP equate in the ACPDBE macro. See *TPFDF Installation and Customization* for more information about the ACPDBE macro.
- You cannot use the `dfdsp` function in a commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

The following example displays a message consisting of all the LRECs from a subfile pointed to in file_ptr. The subfile is released after use.

```
dft_fil *file_ptr;
   .
   .
   .
dfdsp(file_ptr, DFDSP_RELFC);
```

Related Functions

“dfcls–Close a Subfile” on page 86.

dffrl–Ensure an ECB Data Level Is Free

Use this function to free an entry control block (ECB) data level. You can specify the following:

- A specific data level as a number (0x00 to 0x15)
- An open subfile, which frees the data level held by that subfile
- All levels held by SW00SR references.

Format

```
void dffrl_lev(dft_lvl lev);
```

lev
is the data level you want to free, which can be in the range D0–DF.

Entry Requirements

None.

Normal Return

The specified data level is freed.

Error Return

The error indicators in the SW00RTN field of the SW00SR slot have no meaning for this function.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- If you are processing traditional (P-type) files together with TPFDF files, you cannot be sure whether the required ECB levels are free. To avoid conflict, you can ensure that a required ECB level is free by specifying a `dffrl` function before calling a program or function that uses a specific data level.

Examples

The following example frees level DB.

```
dffrl_lev(DB);
```

Related Functions

None.

dfidx–Create an Index Reference

Use this group of functions to create one or more index references to a subfile identified by an algorithm parameter. You can choose to index one or more paths.

Format

```
void dfidx_alg(dft_fil *file, dft_opt options, dft_alg *alg);
void dfidx_alg_pth(dft_fil *file, dft_opt options, dft_alg *alg,
     dft_pth pth);
```

Options Parameter Values:

alg
is a pointer to an algorithm argument that identifies the subfile.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use one of the following values:

DFIDX_PATH_ALL
indexes all paths. Do not use this value when you specify the *pth* parameter.

0 specifies that you do not want to use any processing options.

pth
is the path number for a detail subfile using index support. The value is defined in the DBDEF macro and is a decimal number (0, 1, 2, and so on). The default path number is 0.

See *TPFDF Database Administration* for more information about path indexes.

Entry Requirements

- You must have a detail file available.

- Ensure that the relationship of the index file (or index files, if there are multilevel indexes) to the detail file has been defined with the DBDEF macro by your database administrator.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- Set the *options* parameter to zero if you do not require any processing options.
- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- The TPFDF product determines the amount of data to move in an index LREC by calculating the number of bytes between labels *xxxx*E*yy* and *xxxx*A*yy* in the DSECT for the index file, where *xxxx* is the first 4 characters in the name of the DSECT and *yy* is the primary key.
- Path 0 is the default path. The `dfidx` function creates index references for this path unless you have set one or more different paths using the `dfopt` function or specified the DFIDX_PATH_ALL value of the *options* parameter.
- No actual index structure needs to exist before you index the subfile. All that you need is an existing index file at the highest level of the index. This must be a fixed file. If there is no existing index structure, the TPFDF product creates the required index structure automatically when you call the `dfidx` function.
- If you index a subfile with the `dfidx` function in an application program, you must also remove the index when needed. (You can do this by using the `dfdix` function.)
- When running in detac mode, if an application program creates a pool file using the `dfcre` function and an index reference using the `dfidx` function, the application program must delete the index reference using the `dfdix` function before using the DFCLS_ABORT option on the `dfcls` function. If the index reference is not deleted, subsequent recoup processing may identify the index reference as a broken chain.

Examples

The following example creates an index reference to a subfile (path 1 only). The program provides the algorithm argument in member_number.

```
dft_fil *file_ptr;
char member_number[10];
   .
   .
   .
dfidx_alg_pth(file_ptr, 0, member_number, 1);
```

Related Functions

- “dfadd–Add a Logical Record to a Subfile” on page 73
- “dfcre–Create a Subfile” on page 95
- “dfdix–Delete Index References to a Subfile” on page 105
- “dfopn–Open a Subfile” on page 125.

dfifb–Check a SW00SR Slot

Use this group of functions to check if a SW00SR slot exists. If the slot exists, the function that was called returns the base address of the SW00SR slot.

You can use these functions to test if a particular subfile is open.

Format

```
dft_fil *dfifb_fst();
dft_fil *dfifb_nxt();
dft_fil *dfifb_ref(dft_ref *ref);
dft_fil *dfifb_ref_new(dft_ref *ref, dft_ref *new);
```

new
is a pointer to the character string you want to use as the new file reference name.

ref is a pointer to the reference name of a subfile.

Entry Requirements

None.

Normal Return

The address of the SW00SR slot.

Error Return

NULL pointer.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Use the `dfifb_fst` function to get the first SW00SR slot with an open subfile and then find what other subfiles were opened in sequence by calling the `dfifb_nxt` function.

Examples

- The following example checks if a subfile has been opened.

  ```
  if(dfifb_ref("GR95SR  ") != NULL)    /* then subfile GR95SR is open */
  ```

- The following example returns a pointer to the SW00SR slot of the first open subfile.

  ```
  file_ptr = dfifb_fst();
  ```

Related Functions

“dfopn–Open a Subfile” on page 125.

dfkey–Activate a Key List

Use this function to activate a key list that is used by subsequent functions that access the specified file. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about keys.

Format

```
void dfkey(dft_fil *file,
           dft_kyl *key_list)
void dfkey_nbr(dft_fil *file,
           dft_kyl *key_list, short int n)
```

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

key_list
is a pointer to the key list (SW01SR).

n is the number of keys (1–180) that you have set up in the key list.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- To avoid the possibility of corrupting the organization of a subfile, do not use keys (set up with a `dfkey` function) when adding LRECs to a file. Instead, have the database administrator set up keys in the database definition (DBDEF) that defines the file. These are called *default keys* and override any keys you set up with the `dfkey` function. See *TPFDF Database Administration* for more information about defining default keys.
- You can define any number of key list structures in your program. Each key list can have from 1–180 keys. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

Examples

See “Setting up a Key List with Less than Six Keys” on page 29 and “Setting up a Key List in the Range 1-180” on page 30 for examples of how to set up a key list.

Related Functions

- “dfadd–Add a Logical Record to a Subfile” on page 73
- “dfdel–Delete One or More Logical Records” on page 97
- “dfdsp–Display Logical Records from a Subfile” on page 107
- “dfmrg–Merge Logical Records from Two Subfiles” on page 121

- “dfopn–Open a Subfile” on page 125
- “dfred–Read a Logical Record” on page 134
- “dfsrt–Sort a Subfile” on page 157.

dfmod–Perform or Indicate Logical Record Modifications

Use this group of functions to:

- Indicate that the current logical record (LREC) has been modified
- Modify all LRECs in a file or subfile that match previously established keys.

Format

```
dft_rec *dfmod(dft_fil *file);

dft_rec *dfmod_all(dft_file file, dft_kyl *mod_list)

dft_rec *dfmod_all_opt(dft_file file, dft_opt options,
         dft_kyl *mod_list);

dft_rec *dfmod_all_acc(dft_file file, dft_opt access,
         dft_opt options, dft_kyl *mod_list, dft_xxx acc);

dft_rec *dfmod_all_key(dft_file file, dft_opt options,
         dft_kyl *mod_list, dft_kyl *key_list);

dft_rec *dfmod_all_acc_key(dft_file file, dft_opt access,
         dft_opt options, dft_kyl *mod_list, dft_xxx acc,
         dft_kyl *key_list);
```

Access Parameter Values:

Options Parameter Values:

acc
: is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
: is the method you want to use to access the subfile. Use one of the following values:

  DFMOD_ALG
  : specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of

algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFMOD_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFMOD_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

key_list
is a pointer to the selection key list that determines which LRECs are to be modified. See “df_setkey–Setting Up a Key in a Key List” on page 150 for information about how to set up a selection key list.

mod_list
is a pointer to the modification key list that describes how the LRECs are to be modified. See “df_setkey–Setting Up a Key in a Key List” on page 150 for information about how to set up a modification key list.

options
are the processing options for this function. Use the following values:

DFMOD_BEGIN
modifies LRECs starting from the beginning of the subfile. If you do not specify DFMOD_BEGIN, LRECs are modified starting from the current LREC.

DFMOD_FULLFILE
modifies LRECs in all subfiles of the file, not just the current subfile.

DFMOD_NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this function:
- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the DFMOD_NODUMP value is not recommended because it can prevent system errors from being issued that indicate a critical problem.

DFMOD_NOKEY
deactivates any currently active keys.

0 specifies that you do not want to use any processing options.

Entry Requirements

None.

Normal Return

- If you are using the `dfmod` functions to indicate that you have modified a record in storage, a pointer to the current LREC will be returned.
- If a global modification is being done, the value returned is 0 and SW00RTN will contain a value of DFC_RCDNF or DFC_EOF. This is a normal return condition.

Error Return

None.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- If you locate an LREC or header in a subfile using the `dfred` function and modify the LREC data using assembler instructions, you must ensure that the changes are recorded on DASD. Use the `dfmod` function to do this.

  The `dfmod` function sets an indicator in the block to say that it has been changed. The TPFDF product writes this block to DASD when you close or checkpoint the subfile. You must call the `dfmod` function while the LREC you modified is still current. If you allow the program to read other LRECs in the subfile before you call the `dfmod` function, some modifications to LRECs can be lost.

  Attention: Do not use the `dfmod` functions if you have changed:
  - The size of the existing LREC
  - Any key fields
  - Any fields in the LREC that are also used as index key fields.

  Instead, delete the old LREC with a `dfdel` function and add a new LREC with a `dfadd` function.
- Use one of the `dfmod_all` functions to perform global modifications. Global modification allows you to update multiple LRECs with a single `dfmod` function call. You must provide the modification key list with the *modlist* parameter. The modification key list contains the rules for updating the LRECs. See “Setting Up and Using a Key List” on page 26 for more information about defining a modification key list.
- If you use global modification (the `dfmod_all` functions) when KEYCHECK=YES is defined on the DBDEF macro and any of the fields being modified overlap any default key fields for that primary key in the file, the TPFDF product issues a system error and processing ends. All records that were changed before processing ended remain changed.

Examples

- The following example shows a global modification that uses an active key list to select the records to be modified. The code does this by setting up and activating a selection key list containing a primary key and two additional keys. A modification key list is then established. The modification key list indicates that two different fields in the LREC will be modified by adding a halfword value and replacing a single byte character value respectively. The `dfmod_all` function is then called to modify all LRECs that match the selection key list criteria.

```
/* set up the search keys */
df_nbrkeys(&select_keylist,3);
df_setkey(&select_keylist,1,offsetof(proto_lrec,lrec_id),
          member_size(proto_lrec,lrec_id),DF_EQ,NULL,X'80',
          DF_UPORG,DF_CONST);
df_setkey(&select_keylist,2,offsetof(proto_lrec,key1),
          member_size(proto_lrec,key1),DF_EQ,search_key1,0,
          DF_UPORG,DF_CHAR);
df_setkey(&select_keylist,3,offsetof(proto_lrec,key2),
          member_size(proto_lrec,key2),DF_EQ,search_key2,0,
          DF_UPORG,DF_CHAR);

/* activate the search key list */
dfkey(file_ptr,&select_keylist);

/* set up the modify keys */
df_nbrkeys(&mod_keylist,2);
df_setkey_mod(&mod_keylist,1,offsetof(proto_lrec,field1),
              member_size(proto_lrec,field1),modify_val1,0,
              DF_ADD_SHORT);
df_setkey_mod(&mod_keylist,2,offsetof(proto_lrec,field2),
              member_size(proto_lrec,field2),0,modify_val2,
              DF_MVI);

/* Modify all LRECs matching the established criteria */
dfmod_all_opt(file_ptr,DFMOD_BEGIN,&mod_keylist);
```

- The following is another example of global modification in C. In this example, because no key list is active, all LRECs from LREC number 5 to the end of the file will be modified. This example also shows how a global modification can be done without using the `df_setkey_mod` function.

```
lrec_ptr = dfred_nbr(file,DFRED_LRECNBR, 0,5);

memset(&mod_keylist, 0x00,74);
mod_keylist.sw01nky = 1;
mod_keylist.kit[0].sw01dis = 8;
mod_keylist.kit[0].sw01len = 2;
mod_keylist.kit[0].sw01sea = empname;
mod_keylist.kit[0].sw01id2 = DF_MVC;

lrec_ptr = dfmod_all(file,&mod_keylist);
```

Related Functions

- “dfadd–Add a Logical Record to a Subfile” on page 73
- “dfrep–Replace a Logical Record with Another Logical Record” on page 143.

dfmrg–Merge Logical Records from Two Subfiles

Use this function to merge two subfiles into one subfile.

Format

```
void dfmrg(dft_fil *file, dft_fil *input,
     dft_opt options, dft_kyl *key_list);
```

Options Parameter Values:

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

input
is a pointer to the SW00SR slot of the input subfile that will be merged into the subfile referenced by *file*.

key_list
is a pointer to the key list of the merged output file. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

options
are the processing options for this function. Use the following values:

DFMRG_FULLFILE
merges LRECs from the entire input file to the output subfile referenced by *file*.

DFMRG_RELEASE
releases the SW00SR slot of the input file after processing the merge.

DFMRG_RELFC
releases the input subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

0 specifies that you do not want to use any processing options.

Entry Requirements

You must open both subfiles before calling `dfmrg`.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent functions when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with functions that access or update records without using fullfile processing.
- Make sure the input file contains LRECs that are in the same order as the output file. If the input file is not in the same order as the output file, do not merge the files until they are in the same order. Use the `dfsrt` function to arrange the input file in the same order as the output file.
- The keys that you specify with this function are used to sort the LRECs in the output subfile unless the file is a B+Tree file. The *keylist* parameter is ignored for B+Tree files. The output file is organized according default keys defined on the DBDEF macro for the file. See *TPFDF Database Administration* for more information about default keys.
- The output file cannot be in detac mode when you use the `dfmrg` function.
- You cannot call additional TPFDF functions to the input file until the file is closed if the following conditions are true:
  – You specify the DFMRG_FULLFILE option
  – You do not specify the DFMRG_RELEASE option
  – The end-of-file indicator is set.

  However, you can specify the DFCLS_REUSE value on the `dfcls` function. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- When the `dfmrg` function has completed processing, the output subfile is left open and must be closed using the `dfcls` function before the ECB exits. If you specify the DFMRG_RELEASE value, the `dfmrg` function will close the input file.
- You cannot use this function with P-type files.
- Figure 18 shows how the `dfmrg` function merges LRECs from two subfiles.

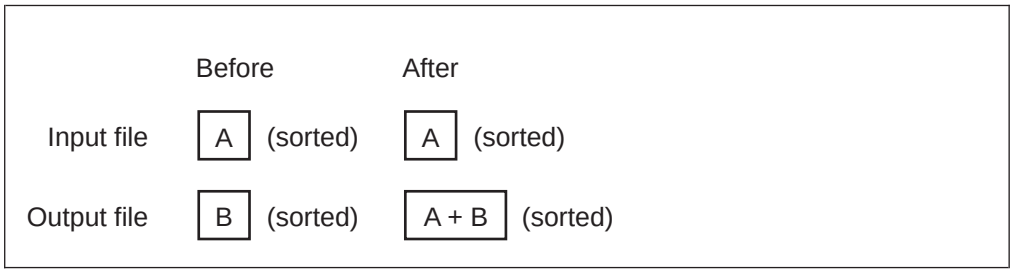


Figure 18. Merging LRECs from Two Subfiles. The input file is defined by the input parameter and the output file is defined by the file parameter.

The merged subfile is in the sequence you specify in the key list pointed to in the *key_list* parameter. The function does not modify the input subfile that you specify with the *input* parameter. The TPFDF product leaves the entry control block (ECB) data level for the input subfile free after use.

- If you use the dfmrg function in a commit scope, open the files or subfiles to be merged in the same commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

The following example merges the two open subfiles referenced by file_ptr and input_ptr. All the LRECs go into the file_ptr subfile.

```
/* first set up keys */

dft_fil *input_ptr;
dft_kyl keys;

df_nbrkeys(&keys, 1);
df_setkey(&keys, 1, offsetoff(struct gr95sr, gr95nam),
          member_size(struct gr95sr, gr95nam), 0, NULL, 0, DF_UPORG, DF_CHAR);

dfmrg(file_ptr, input_ptr, 0, &keys);
```

Related Functions

“dfsrt–Sort a Subfile” on page 157.

df_nbrkeys–Setting Up the Number of Keys

Use this function to set up the number of keys that you want to use in a particular key list structure.

Format

```
void df_nbrkeys(dft_kyl *key_list, short int n);
```

key_list
is a pointer to the key list structure you are setting up.

n is the number of keys (1–180) that you want in the key list.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- You can define any number of key list structures in your program. Each key list can have from 1–180 keys. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

Examples

See “Setting up a Key List with Less than Six Keys” on page 29 and “Setting up a Key List in the Range 1-180” on page 30 for examples of how to set up a key list.

Related Functions

“df_setkey–Setting Up a Key in a Key List” on page 150.

dfopn–Open a Subfile

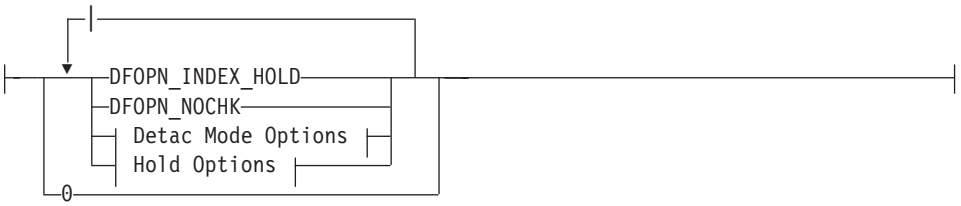
Use this group of functions to open a subfile. This is the first TPFDF function you use with any file. The first `dfopn` function creates a database interface block (DBIFB) and reserves a SW00SR slot in the DBIFB. The SW00SR slot contains control information about the subfile. See *TPFDF General Information* for more information about SW00SR and this control information.

Format

```
dft_fil *dfopn(dft_ref *ref_name, dft_fid *id, dft_opt options);

dft_fil *dfopn_acc(dft_ref *ref_name, dft_fid *id,
        dft_opt access, dft_opt options, dft_xxx acc);

dft_fil *dfopn_spa(dft_ref *ref_name, dft_fid *id, dft_opt options,
        dft_spc spc, dft_sps sps);

dft_fil *dfopn_are(dft_ref *ref_name, dft_fid *id, dft_opt options,
        dft_are *are);

dft_fil *dfopn_tpn(dft_ref *ref_name, dft_fid *id, dft_opt options,
        dft_tpn tpn);

dft_fil *dfopn_acc_spa(dft_ref *ref_name, dft_fid *id, dft_opt access,
        dft_opt options, dft_xxx acc, dft_spc spc,dft_sps sps);

dft_fil *dfopn_acc_are(dft_ref *ref_name, dft_fid *id, dft_opt access,
        dft_opt options, dft_xxx acc,dft_are *are);

dft_fil *dfopn_acc_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt access, dft_opt options, dft_xxx acc, dft_tpn tpn);

dft_fil *dfopn_spa_are(dft_ref *ref_name, dft_fid *id,
        dft_opt options, dft_spc spc, dft_sps sps,dft_are *are);

dft_fil *dfopn_spa_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt options, dft_spc spc, dft_sps sps, dft_tpn tpn);

dft_fil *dfopn_are_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt options, dft_are *are, dft_tpn tpn);

dft_fil *dfopn_acc_spa_are(dft_ref *ref_name, dft_fid *id,
        dft_opt access, dft_opt options, dft_xxx acc, dft_spc spc,
        dft_sps sps, dft_are *are);

dft_fil *dfopn_acc_spa_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt access, dft_opt options,
        dft_xxx acc, dft_spc spc, dft_sps sps, dft_tpn tpn);

dft_fil *dfopn_acc_are_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt access, dft_opt options,
        dft_xxx acc, dft_are *are, dft_tpn tpn);

dft_fil *dfopn_spa_are_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt options, dft_spc spc,
        dft_sps sps, dft_are *are, dft_tpn tpn);

dft_fil *dfopn_acc_spa_are_tpn(dft_ref *ref_name, dft_fid *id,
        dft_opt access, dft_opt options, dft_xxx acc,
        dft_spc spc, dft_sps sps, dft_are *are, dft_tpn tpn);
```

Access Parameter Values:

Options Parameter Values:



Detac Mode Options:

Hold Options:

acc
is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
is the method you want to use to access the subfile. Use one of the following values:

DFOPN_ALG
specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFOPN_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFOPN_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

are
is the address of an area where the user information is loaded from index LRECs on subsequent `dfred` function calls. The data that will be provided is specified on the DBDEF macro for the index file.

id is a pointer to the 2-byte file identifier (held in each block of the file). You can specify this in character form or as 4 hexadecimal digits.

options
are the processing options for this function. Use the following values:

DFOPN_DETAC
opens the subfile in detac mode. When the subfile is in detac mode, all modified blocks are saved in main storage. No blocks are written to DASD until you checkpoint the subfile (using the `dfckp` function) or close the subfile (using the `dfcls` function).

If you do not want to keep any modifications that you made to the subfile opened with DFOPN_DETAC, you can use the DFCLS_ABORT value for the *options* parameter of the `dfcls` function.

Note: The TPF system and the ALCS environment issues a 000010 system error if an application program does not give up control in the time allotted by the application time-out counter. When processing in detac mode, a TPFDF application program can require more than the allotted time on a database with a large data structure. To prevent the 000010 system error, you can change the setting of the &TPFDBDV symbol in the DBLCL macro.

See *TPFDF Installation and Customization* for more information about the &TPFDBDV symbol and the DBLCL macro.

DFOPN_NODET
specifies that you do not want the subfile opened in detac mode.

DFOPN_HOLD
potentially holds the subfile that you are accessing and prevents two or more application programs from modifying the subfile at the same time. Holding occurs on the following TPFDF call that accesses the subfile if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the subfile will not occur until the subfile is no longer held. If more than one application can update the same subfile, or the file is processed in fullfile mode, you must specify this value to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, affect hold processing.

DFOPN_NOHOLD
specifies that the TPFDF product does not hold the file blocks and other applications can read or write blocks. (This implies that you are not going to modify the subfile.)

DFOPN_INDEX_HOLD
potentially holds any index files that reference the subfiles you are accessing and prevents two or more application programs from modifying the index files at the same time. Holding occurs on the following TPFDF call that accesses the subfile if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the index file will not occur until the index file is no longer held. If more than one application can update the same index file, or the file is processed in fullfile mode, you must specify this value to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, affect hold processing.

DFOPN_NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks that are read from DASD with subsequent `dfred` function calls. In addition, use this value to prevent the TPFDF product from using a random RCC value when creating new subfiles. If this value is used, new subfiles will be created without an RCC value unless you specify the DFOPT_CHKA value on the `dfopt` function. In this case, new subfiles are created with the RCC specified by the `dfopt` function.

0 specifies that you do not want to use any processing options.

ref_name
is the address of a variable containing the 8-byte reference name of the subfile to be opened.

spc
is the character that you want to fill the space area you have requested.

sps
is the size of the space area you require, which can be a maximum of 3952 bytes.

tpn
is a pointer to a variable containing a 3-character symbolic name of the tape or sequential data set you want to use. The TPFDF product then writes all overflow blocks to tape rather than to DASD.

You cannot open B+Tree files using the *tpn* parameter.

Entry Requirements

Ensure that the subfile you open was previously defined in a DSECT macro and in a DBDEF macro instruction by your database administrator.

Normal Return

The address of the SW00SR slot.

Error Return

The error indicators in the SW00RTN field of the SW00SR slot have no meaning for this function.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_xxx`. See the description of the specific parameter for information about what type definition to use for that parameter.
- You cannot use this function with a T-type file because a T-type file is a temporary logical record (LREC) stored in a W-type file and is not defined in the DBDEF macro. See *TPFDF Database Administration* for more information about T-type files.
- If you use this function with a W-type file, the DFOPN_HOLD value is the default. See *TPFDF Database Administration* for more information about W-type files.
- It is not necessary to specify data level independence (DLI) with this function. The TPFDF product preserves all data levels holding blocks before a macro or function call. See “Data Level Usage” on page 3 for more information about DLI.

Examples

- The following example opens a subfile that has DSECT name GR91SR. The TPFDF product sets a pointer to the SW00SR slot in file_ptr. The file ID is "S0".

```
dft_fil *file_ptr;
file_ptr = dfopn("GR91SR  ", "S0", DFOPN_HOLD | DFOPN_DETAC);
```

- The following example opens a subfile that has a DSECT name of GR91SR and requests 50 bytes of space. The file ID is X'C701'.

```
#define file_id "\xC7\x01"
file_ptr = dfopn_spa("GR91SR  ", file_id, DFOPN_HOLD, ' ', 50);
```

- The following example opens a subfile that has a DSECT name of GR91SR in detac mode. The program provides a file address in *faddr*. The file ID is "S0".

```
file_ptr = dfopn_acc("GR91SR  ", "S0", DFOPN_FADDR, DFOPN_HOLD, faddr);
```

Related Functions

“dfcls–Close a Subfile” on page 86.

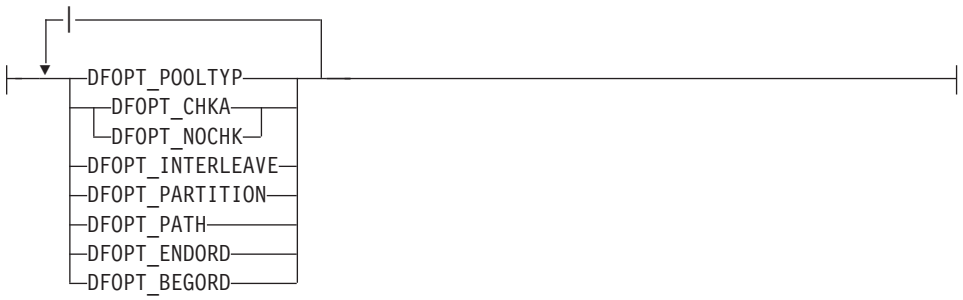
dfopt–Set Optional Information

Use this group of functions to set options after opening a file.

Format

```
void dfopt1(dft_fil *file, dft_opt options, dft_xxx opt1);
void dfopt2(dft_fil *file, dft_opt options, dft_xxx opt1,
     dft_xxx opt2);
void dfopt3(dft_fil *file, dft_opt options, dft_xxx opt1,
     dft_xxx opt2, dft_xxx opt3);
void dfopt4(dft_fil *file, dft_opt options, dft_xxx opt1,
     dft_xxx opt2, dft_xxx opt3, dft_xxx opt4);
void dfopt5(dft_fil *file, dft_opt options, dft_xxx opt1,
     dft_xxx opt2, dft_xxx opt3, dft_xxx opt4, dft_xxx opt5);
void dfopt6(dft_fil *file, dft_opt options, dft_xxx opt1,
     dft_xxx opt2, dft_xxx opt3, dft_xxx opt4, dft_xxx opt5,
     dft_xxx opt6);
```

Options Parameter Values:



file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. The values you specify indicate what additional parameters (such as, *opt1*, *opt2*, and so on) you are supplying with the function call.

The number of values you must specify for the *options* parameter is based on which `dfopt` function you use. For example, if you are using `dfopt1`, specify one of the following values; if you are using `dfopt2`, specify two of the following values; and so on.

DFOPT_POOLTYP
indicates that you are specifying the pool type with the *pooltype* parameter. Use this value to override the pool type defined in the DSECT for the subfile.

DFOPT_CHKA
indicates that you are specifying a record code check (RCC) character with the *chka* parameter. Use this value when you want to check the RCC value

of the prime block of the subfile that it opens with `dfopn` and any blocks that it reads using `dfred`. If the RCC values do not match, the TPFDF product returns an error condition.

DFOPT_NOCHK
indicates that you are specifying a record code check (RCC) character with the *chka* parameter. Use this value when you do not want to check the record code check (RCC) value of blocks that are read with subsequent `dfred` functions.

DFOPT_INTERLEAVE
indicates that you are specifying an interleave number with the *partition* parameter and you want to access LRECs in this interleave.

DFOPT_PARTITION
indicates that you are specifying a partition number with the *partition* parameter and you want to access LRECs in this partition.

DFOPT_PATH
indicates that you are specifying a path number with the *path* parameter. Use this value if you are opening an indexed subfile that has more than one path. See *TPFDF Database Administration* for more information about index paths.

DFOPT_ENDORD
indicates that you are specifying an ordinal number for the end of the subfile with the **endord** parameter.

DFOPT_BEGORD
indicates that you are specifying an ordinal number for the start of the subfile with the *begord* parameter.

optn
are the optional parameters (such as *opt1*, *opt2*, and so on) that you are specifying. Replace the *optn* parameters with any of the following parameters.

pooltype
is the pool type, which can be one of the following:

0 uses the pool type defined by the PF0 parameter of the DBDEF macro.

1 uses the pool type defined by the PF1 parameter of the DBDEF macro.

2 uses the pool type defined by the PF2 parameter of the DBDEF macro.

Use this parameter if you specified the DFOPT_POOLTYP value with the *options* parameter.

chka
is a record code check (RCC) value in the range of 0x00–0xFF. Use this parameter if you specified the DFOPT_CHKA or DFOPT_NOCHK value with the *options* parameter.

partition
is one of the following:

num
is a partition or interleave number, which you can get from your database administrator.

DFOPT_ALL
specifies all partitions or all interleaves of the file.

Use this parameter if you specified the DFOPT_PARTITION or DFOPT_INTERLEAVE value with the *options* parameter.

path
is the path number, which is a decimal value defined with the DBDEF macro, that you want to use to access detail subfiles. See your database administrator for this path value and see *TPFDF Database Administration* for more information about index paths.

Use this parameter if you specified the DFOPT_PATH value with the *options* parameter.

begord
is the beginning ordinal number. Use this parameter if you specified the DFOPT_BEGORD value with the *options* parameter.

endord
is the ending ordinal number. Use this parameter if you specified the DFOPT_ENDORD value with the *options* parameter.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_xxx`. See the description of the specific parameter for information about what type definition to use for that parameter.
- Any individual parameter (for example, *partition*) may or may not be present when you call a `dfopt` function. However, you must specify the appropriate number of optional parameters in total. For example, if you call `dfopt4`, you must specify a total of 4 optional parameters.
- Always specify the optional parameters that are present in the order shown, omitting the ones that you do not want to use.
- See your database administrator for more information about using these parameters.

Examples

- The following example sets the partition number to 2 and the path number to 1.

```
dft_fil *file_ptr;
   .
   .
   .
dfopt2(file_ptr, DFOPT_PARTITION | DFOPT_PATH, 2, 1);
```

- The following example sets begin ordinal to 100, pool type to 1, and interleave number to 3. (The parameters must be in the order shown.)

```
dft_fil *file_ptr;
   .
   .
   .
dfopt3(file_ptr, DFOPT_POOLTYPE | DFOPT_PARTITION | DFOPT_BEGORD, 1, 3, 100);
```

Related Functions

- “dfadr–Provide the File Address of a Prime Block” on page 80
- “dfcre–Create a Subfile” on page 95
- “dfdix–Delete Index References to a Subfile” on page 105
- “dfidx–Create an Index Reference” on page 112
- “dfopn–Open a Subfile” on page 125.

dfred–Read a Logical Record

Use this group of functions to read a logical record (LREC) or block header and get the address where the record is stored. You can read the next LREC in sequence or specify details of the LREC you require.

You can also use these functions to read a sequence of LRECs. In this case, you perform a sequence of `dfred` calls and get a different LREC each time.

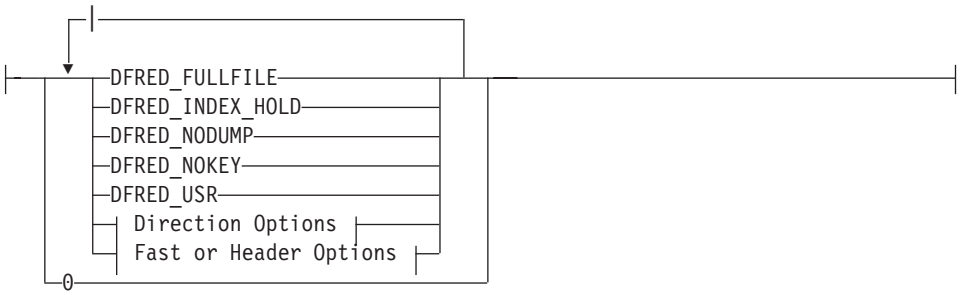
Format

```
dft_rec *dfred(dft_fil *file, dft_opt options);

dft_rec *dfred_acc(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc);

dft_rec *dfred_nbr(dft_fil *file, dft_opt nbr_type,
        dft_opt options, dft_xxx nbr);

dft_rec *dfred_are(dft_fil *file, dft_opt options, dft_are *are);

dft_rec *dfred_pth(dft_fil *file, dft_opt options, dft_pth pth);

dft_rec *dfred_acc_nbr(dft_fil *file, dft_opt nbr_type, dft_opt access,
        dft_opt options, dft_xxx acc, dft_xxx nbr);

dft_rec *dfred_acc_are(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc, dft_are *are);

dft_rec *dfred_acc_pth(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc, dft_pth pth);

dft_rec *dfred_nbr_are(dft_fil *file, dft_opt nbr_type, dft_opt options,
        dft_nbr nbr, dft_are *are);

dft_rec *dfred_nbr_pth(dft_fil *file, dft_opt nbr_type, dft_opt options,
        dft_xxx nbr, dft_pth pth);

dft_rec *dfred_are_pth(dft_fil *file, dft_opt options, dft_are *are,
        dft_pth pth);

dft_rec *dfred_acc_nbr_are(dft_fil *file, dft_opt nbr_type, dft_opt access,
        dft_opt options, dft_xxx acc, dft_xxx nbr, dft_are *are);

dft_rec *dfred_acc_nbr_pth(dft_fil *file, dft_opt nbr_type, dft_opt access,
        dft_opt options, dft_xxx acc, dft_xxx nbr, dft_pth pth);

dft_rec *dfred_acc_are_pth(dft_fil *file, dft_opt access, dft_opt options,
        dft_xxx acc, dft_are *are, dft_pth pth);

dft_rec *dfred_nbr_are_pth(dft_fil *file, dft_opt nbr_type, dft_opt options,
        dft_xxx nbr, dft_are *are, dft_pth pth);

dft_rec *dfred_acc_nbr_are_pth(dft_fil *file, dft_opt nbr_type,
        dft_opt access, dft_opt options, dft_xxx acc, dft_xxx nbr,
        dft_are *are, dft_pth pth);
```

Access Parameter Values:

Nbr_Type Parameter Values:

Options Parameter Values:



Direction Options:

Fast or Header Options:

acc
is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
is the method you want to use to access the subfile. Use one of the following values:

DFRED_ALG
specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFRED_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFRED_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

are
is a pointer to an area to which the TPFDF product copies user data from the index LREC referencing the detail subfile you are accessing. This user data is defined in the DBDEF macro of the detail subfile. See *TPFDF Database Administration* for more information about defining this user data.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

nbr
is a variable containing one of the following based on the value of the *nbr_type* parameter:

- A single LREC number (type `dft_nbr`) if the *nbr_type* parameter is set to DFRED_LRECNBR.
- A pointer to a list of LRECs (type `dft_rcl`) if the *nbr_type* parameter is set to DFRED_LIST. Specify the LRECs in this list as character array. For example:

  ```
  char nbrs[10] = "2/3-6/LAST";
  ```

  The list contains one or more LREC sequence numbers separated by a slash (/). You can also specify a range of sequence numbers by separating the beginning and end of the range by a hyphen (-). You can use LAST to mean the last LREC of the subfile and ALL to mean all the remaining LRECs. You can also end the list with a nonnumeric character.

  Notes:

  1. The ranges must be in ascending order; if one is found out of order, that range and all subsequent ranges are ignored.

     For example, if there are 41 LRECs in a subfile, the following lists all have the same effect:

```
20/31/32/33/37/38/39/40/41
20/31/32/33/37-41
20/31-33/37-LAST
20/31-33/37/ALL
```

2. You ***cannot*** specify the number zero in the list of LREC numbers, even if you specify the ADJUST parameter with a value that would adjust the number zero to a valid LREC number.

- A stack reference number (type `dft_srn`) if the *nbr_type* parameter is set to DFRED_STACKREF.
- A pointer to a stack area (type `dft_stk`) if the *nbr_type* parameter is set to DFRED_STACK.

Notes:

1. If you use the #TPFDB0D algorithm, you must specify a specific LREC number for the value of *nbr*.
2. LRECs are numbered in increasing order from the start of the subfile (the first LREC in the prime block has sequence number 1).
3. If you specify this parameter with active keys, only those LRECs that match the key conditions are included in the sequence numbering; LRECs that do not match are ignored.
4. In functions that do not include the *nbr_type* parameter, the value of *nbr* is a specific LREC number.

nbr_type
is one of the following:

DFRED_LIST
specifies that you are supplying a pointer to a list of LREC numbers (type `dft_rcl`) in the *nbr* parameter. The TPFDF product reads these LRECs in sequence on subsequent `dfred` function calls (unless you change the `dfred` parameters).

DFRED_LRECNBR
specifies that you are supplying the sequence number of a single LREC (type `dft_nbr`) in the *nbr* parameter. The TPFDF product reads the LREC with this sequence number.

DFRED_STACK
specifies that you are supplying a pointer to a stack area (type `dft_stk`) in the *nbr* parameter.

DFRED_STACKREF
specifies that you are supplying a stack reference number (type `dft_srn`) in the *nbr* parameter.

The DFRED_STACK and DFRED_STACKREF values allow you to read an LREC that you have retained with an earlier `dfret` function.

options
are the processing options for this function. Use the following values:

DFRED_BACKWARD
reads backward through the subfile; that is, the TPFDF product reads the LREC immediately before the current LREC position.

Notes:

1. If you use DFRED_BACKWARD and also use the `dfret` function, you must specify DFRET_STACK or DFRET_STACKREF with the `dfret`

function. See “dfret–Retain a Logical Record Position” on page 145 for more information about these values and the `dfret` function.

2. If you use DFRED_BACKWARD, the default and recommended setting for symbol &DB013A in the DBLCL macro is 0. This setting requires files to use full backward chaining (bit 0 of &SW00OP1 is set) to read backward. See *TPFDF Database Administration* for more information about defining full backward chaining. See *TPFDF Installation and Customization* for more information about the DBLCL macro.

   Exception: If the file uses add current processing (bit 2 of &SW00OP1 is set) with no chains (&SW00NOC = 0), you can specify DFRED_BACKWARD regardless of how bit 0 of &SW00OP1 is set.

DFRED_BEGIN
reads LRECs from the start of the subfile.

DFRED_LAST
reads the last LREC from the subfile. If you have specified keys with the read, the TPFDF product reads the last LREC that matches the keys.

DFRED_PREVIOUS
reads the LREC saved with the last `dfret` function that used a DFRET_CURRENT value for the *ret_type* parameter.

DFRED_FAST
used for migration purposes only; use the DFRED_INLINE or DFRED_NOKEY values instead. If you specify this value, the DFRED_NOKEY value is implemented; that is, keys that are currently active are deactivated.

DFRED_HEADER
locates the subfile header in the prime block and returns the address in field SW00RC rather than the address of an LREC.

If you specify the DFRED_HEADER and DFRED_FULLFILE values on an open subfile, the `dfred` function retrieves the header of the next subfile.

DFRED_INLINE
provides inline processing for this function. You can only use this value if you are reading LRECs sequentially, without any key parameters. This option also deactivates any keys that are currently active; that is, any previous key arguments are set to zero.

DFRED_FULLFILE
reads an LREC from any of the subfiles in the file. If you do not specify this value, you can only read an LREC in the current subfile.

DFRED_INDEX_HOLD
potentially holds any index files that reference the subfiles you are accessing and prevents two or more application programs from modifying the index files at the same time. Holding occurs if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the index file will not occur until the index file is no longer held. If more than one application can update the same index file, or the file is processed in fullfile mode, you must specify this value to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, affect hold processing.

DFRED_NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this function:
- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the DFRED_NODUMP value is not recommended because it can prevent system errors from being issued that indicate a critical problem.

DFRED_NOKEY
deactivates any currently active keys.

DFRED_USR
returns a pointer to the userLREC. Use this option when working with extended LRECs. See “Using Extended Logical Records” on page 4 for more information about extended LRECs.

0 specifies that you do not want to use any processing options.

pth
is the path number for a detail subfile using index support. The value is defined in the DBDEF macro and is a decimal number (0, 1, 2, and so on). The default path number is 0.

See *TPFDF Database Administration* for more information about path indexes.

Entry Requirements

If you specify the DFRED_PREVIOUS, DFRED_STACK, or DFRED_STACKREF values, you must first open the subfile using the DFOPN_DETAC or DFOPN_HOLD values for the *options* parameter on the `dfopn` function.

Normal Return

- If you specify the DFRED_USR value, a pointer to the userLREC portion of the extended LREC is returned.
- If you specify the DFRED_HEADER value, a pointer to the standard header of the block is returned.
- In all other cases, a pointer to the LREC that is read is returned.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.

- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_xxx`. See the description of the specific parameter for information about what type definition to use for that parameter.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent functions when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with functions that access or update records without using fullfile processing.

  Note: The DFRED_HEADER value can be used with the DFRED_FULLFILE value to move from subfile to subfile, which can then be accessed using the DBRED macros that do not use the DFRED_FULLFILE value.
- If you have set up keys with the file you are reading (using the `dfkey` function), the TPFDF product ignores LRECs where the LREC key does not match the keys.
- If you specify the DFRED_FULLFILE option, and the end-of-file indicator is set, you cannot issue additional TPFDF functions until the file is closed. However, you can specify the DFCLS_REUSE option on the `dfcls` function. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- Any active keys are ignored when you use the #TPFDB0D algorithm.
- To ensure an LREC is retrieved accurately when you use the `dfred` function, do not use the `dfret` function with the DFRET_STACK and DFRET_STACKREF values specified on the same open file. If you do, the wrong LREC could be retrieved. You must close and reopen the file each time you alternate between specifying the DFRET_STACK and DFRET_STACKREF values.

Examples

- The following example reads an LREC from the current subfile. The new LREC goes into lrec_ptr.

```
dft_fil *file_ptr;
struct gr95sr *lrec_ptr;
   ⋮
lrec_ptr = dfred(file_ptr, 0);
```

- The following example reads an LREC retained from a subfile. The subfile is specified with an algorithm. (The program has set up a pointer to the algorithm argument in member_number.)

```
dft_fil *file_ptr;
char member_number[10];
   ⋮
lrec_ptr = dfred_acc(file_ptr, DFRED_ALG, DFRED_PREVIOUS, member_number);
```

- The following example shows how to read a file using keys, without using default keys.

```
struct gr95sr lrec_def;
struct gr95sr *lrec_ptr;
dft_kyl select_keylist;

df_nbrkeys(&select_keylist,4);
df_setkey(&select_keylist,1,offsetof(lrec_def,lrec_id),
          member_size(lrec_def,lrec_id),DF_EQ,NULL,X'80',
          DF_UPORG,DF_CONST);
df_setkey(&select_keylist,2,offsetof(lrec_def,key1),
          member_size(lrec_def,key1),DF_EQ,search_name,0,
          DF_UPORG,DF_CHAR);
df_setkey(&select_keylist,3,offsetof(lrec_def,key2),
```

```
          member_size(lrec_def,key2),DF_EQ,search_city,0,
          DF_UPORG,DF_CHAR);
df_setkey(&select_keylist,4,offsetof(lrec_def,key3),
          member_size(lrec_def,key3),DF_EQ,search_sal,0,
          DF_UPORG,DF_CHAR);
dfkey(file_ptr,&select_keylist);
lrec_ptr = dfred(file_ptr, 0);
```

- The following example shows how to read a file using read-only and read-write default keys. After the prototype LREC is initialized to zeros, each of the selection key values (including the primary key) is copied into the prototype LREC. A call to the `df_setkey_dbdef` function sets up the selection key list structure and a call to the `dfkey` function copies the default key values from the key list into the SW00SR. Finally, a call to the `dfred` function returns the next LREC that matches the default keys. See “df_setkey–Setting Up a Key in a Key List” on page 150 for more information.

  ```
  struct gr95sr proto_lrec;
  dft_kyl select_keylist;

  memset(&proto_lrec, 0x00, sizeof(proto_lrec));
  proto_lrec.lrec_id = 0x80;
  memcpy(proto_lrec.key1,search_name,sizeof(proto_lrec.key1));
  memcpy(proto_lrec.key2,search_city,sizeof(proto_lrec.key2));
  memcpy(proto_lrec.key3,search_sal,sizeof(proto_lrec.key3));
  df_setkey_dbdef(&select_keylist,&proto_lrec,0x80);
  dfkey(file_ptr,&select_keylist);
  lrec_ptr = dfred(file_ptr,0);
  df_setkey_dbdef(&select_keylist,&proto_lrec,0x06);
  dfkey(file_ptr,&select_keylist);
  lrec_ptr = dfred(file_ptr,0);
  ```

- The following example shows how to read a file using Boolean logic. It sets up a key list containing three Boolean logic keys before reading the LREC.

  The logic used is equivalent to:

  ```
  gr95key AND (gr95nam OR gr95adr).
  ```

  See “df_setkey–Setting Up a Key in a Key List” on page 150 for more information about setting up a key list for Boolean logic.

  ```
  dft_rec *lrec91;
  dft_pky pky = 0x80;
  dft_pky name1 = "Smith";
  dft_pky addr1 = "Main Street";
  dft_kyl keys;

  /* set up the keys */
  df_nbrkeys(&keys, 3);
  df_setkey_bool(&keys, 1, offsetof(struct gr95sr, gr95key),
                 1, DF_EQ, &pky, 0, DF_UPORG, DF_CHAR, DF_ANDIF);
  df_setkey_bool(&keys, 2, offsetof(struct gr95sr, gr95nam),
                 10, DF_EQ, &name1, 0, DF_UPORG, DF_CHAR, DF_OR);
  df_setkey(&keys, 3, offsetof(struct gr95sr, gr95adr),
            10, DF_EQ, &addr1, 0, DF_UPORG, DF_CHAR);

  /* activate the key list */
  dfkey(file_ptr, &keys);

  /* read an LREC with matching primary key and either matching */
  /* the name or the address.                                   */
  /* (start at the beginning of the subfile)                    */
  lrec91 = dfred(file_ptr, DFRED_BEGIN);
  ```

- The following example shows how to read an LREC from a detail file using index support. First, open the detail file normally, specifying a DSECT name as the first parameter and then use a `dfred` function to read the LREC by providing an index key string as an algorithm argument.

```
dbptr = dfopn ("GR44DF  ",.........)
alg_string="ALDER";
dfred_acc (dbptr,DFRED_ALG,alg_string);
```

Related Functions

- “dfadd–Add a Logical Record to a Subfile” on page 73
- “dfkey–Activate a Key List” on page 115.

dfrep–Replace a Logical Record with Another Logical Record

Use this group of functions to replace the following:

- A previously read logical record (LREC) with a new LREC
- The userLREC in the current extended LREC with a new userLREC
- The subLREC in the current extended LREC with a new subLREC.

Format

```
dft_rec *dfrep(dft_fil *file, dft_rec *rcd);
dft_rec *dfrep_sub(dft_fil *file, dft_rec *sub);
dft_rec *dfrep_usr(dft_fil *file, dft_rec usr);
dft_rec *dfrep_usr_sub(dft_fil *file, dft_rec *sub,
        dft_rec *usr);
```

file
is a pointer to the base address of the SW00SR slot (defined in c$sw00sr.h) of the subfile that you want to access and is returned by the dfopn function.

rcd
is a pointer to the replacement LREC.

sub
is a pointer to the subLREC that is to replace the subLREC already in the extended LREC.

usr
is a pointer to the userLREC that is to replace the userLREC already in the extended LREC.

Entry Requirements

Before using the dfrep_sub, dfrep_usr, or dfrep_usr_sub function, you must use a dfred function to locate the extended LREC in which you want to replace a subLREC or userLREC.

Normal Return

One of the following:

- Pointer to the new LREC.
- Pointer to the extended LREC containing the new subLREC, the new userLREC, or both.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, dft_fil, dft_ref, and dft_kyl) are defined in the c$cdfapi.h header file.
- The dfrep function replaces the current LREC; that is, the dfrep function does not perform an internal dfred function.
- The new LREC can be larger, smaller, or the same size as the old LREC (if you have defined variable-length LRECs in the DSECT).
- If the subfile uses block index support, the TPFDF product automatically updates the block index when you replace an LREC.
- Do not use the dfrep function if you have changed:

  - Any key fields
  - Any fields in the LREC that are also used as index key fields.

  Instead, delete the old LREC with a `dfdel` function and add a new LREC with a `dfadd` function.
- When using a `dfrep` function that allows a *sub* parameter the current extended LREC must contain one (and only one) subLREC. When an extended LREC contains more than one subLREC, use `dfdel_sub` to delete one or more subLRECs then call a `dfadd` function that allows a *sub* parameter to add each new subLREC.

Examples

- The following example replaces an existing LREC with the LREC pointed to by rec_ptr.

```
dft_fil *file_ptr;
   .
   .
   .
(void) dfrep(file_ptr, DFREP_NEWLREC, &rec_ptr);
```

- The following example replaces the userLREC and the (one) subLREC in the current extended LREC with a new userLREC and a new subLREC respectively.

```
dft_fil *file_ptr;
struct zzzzz1 new_userlrec;
struct zzzzz2 new_sublrec;
   .
   .
   .
dfrep_usr_sub(file_ptr, &new_userlrec, &new_sublrec);
```

Related Functions

- “dfadd–Add a Logical Record to a Subfile” on page 73
- “dfdel–Delete One or More Logical Records” on page 97
- “dfmod–Perform or Indicate Logical Record Modifications” on page 117
- “dfred–Read a Logical Record” on page 134.

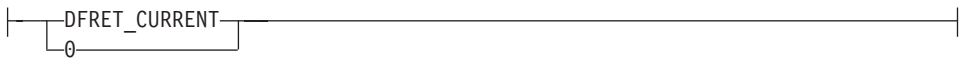
dfret–Retain a Logical Record Position

Use this group of functions to retain the file address and displacement in a block of the current logical record (LREC). You can read the LREC later in your application program by using a `dfred` function with the appropriate *option* parameter value.

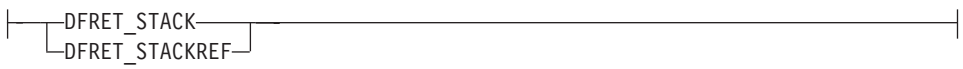
Format

```
void dfret(dft_fil *file, dft_opt options);
void dfret_stk(dft_fil *file, dft_opt stk_type,
     dft_opt options, dft_xxx stk);
```

Options Parameter Values:



Stk_Type Parameter Values:



file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use the following values:

DFRET_CURRENT
specifies that you want to retain the current LREC.

0 specifies that you do not want to use any processing options.

stk
is one of the following based on the value you specify for the *stk_type* parameter:

- A stack reference number (type `dft_srn`) if the *stk_type* parameter is set to DFRET_STACKREF.
- A pointer to a stack area (type `dft_stk`) if the *stk_type* parameter is set to DFRET_STACK.

stk_type
is one of the following:

DFRET_STACK
specifies that you are supplying a pointer to a stack area in the *stk* parameter.

DFRET_STACKREF
specifies that you are supplying a stack reference number in the *stk* parameter.

Both values let you retain a number of LRECs and identify them so that you can later read them with `dfred` functions.

Entry Requirements

Before using this function, you must open the subfile using the DFOPN_DETAC or DFOPN_HOLD values for the *options* parameter of the `dfopn` function.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_`*`xxx`*. See the description of the specific parameter for information about what type definition to use for that parameter.
- To ensure an LREC is retrieved accurately when you use the `dfred` function, do not use the `dfret` function with the DFRET_STACK and DFRET_STACKREF values specified on the same open file. If you do, the wrong LREC could be retrieved. You must close and reopen the file each time you alternate between specifying the DFRET_STACK and DFRET_STACKREF values.

Examples

- The following example retains the current LREC.

```
dft_fil *file_ptr;
   .
   .
   .
dfret(file_ptr, DFRET_CURRENT);
```

- The following example retains the current LREC with a reference number of 100.

```
dft_fil *file_ptr;
   .
   .
   .
dfret_stk(file_ptr, DFRET_STACKREF, DFRET_CURRENT, 100);
```

Related Functions

“dfred–Read a Logical Record” on page 134.

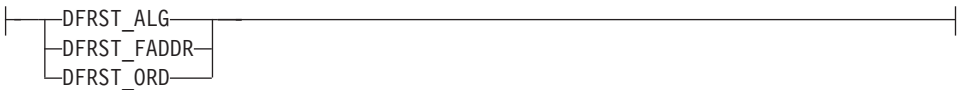
dfrst–Restore a Subfile

Use this group of functions to restore a subfile (previously copied by a `dfcpy` function) to a file address that you specify.

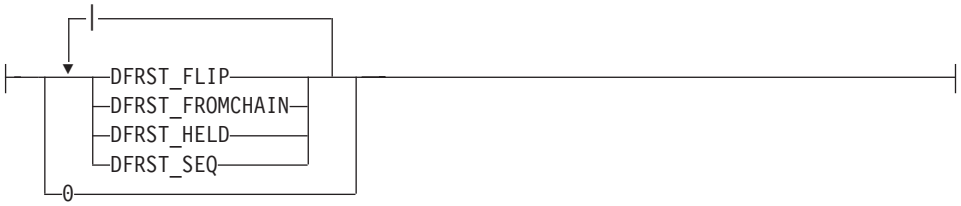
Format

```
void dfrst(dft_fil *file, dft_opt options, dft_fad rstaddr);

void dfrst_acc(dft_fil *file, dft_opt access, dft_opt options,
     dft_fad rstaddr, dft_xxx acc);

void dfrst_seq(dft_fil *file, dft_opt options, dft_fad rstaddr,
     dft_seq seq);

void dfrst_acc_seq(dft_fil *file, dft_opt access, dft_opt options,
      dft_fad rstaddr, dft_xxx acc, dft_seq seq);
```

Access Parameter Values:



Options Parameter Values:



acc
is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
is the method you want to use to access the subfile. Use one of the following values:

DFRST_ALG
specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database*

Administration for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFRST_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFRST_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use the following values:

DFRST_FLIP
specifies to flip the subfiles specified in the *rstaddr* and *file* parameters.

DFRST_FROMCHAIN
restores only the prime block of the subfile. Any overflow blocks in the subfile are chained to this new prime block without copying the overflow blocks to new pool blocks.

DFRST_HELD
restores a subfile that is already held (for example, if you have opened the subfile with a DFOPN_HOLD value).

DFRST_SEQ
specifies that you want to check the sequence number when it restores the subfile. This ensures that if there are several copies of a subfile, you restore the correct one.

If you use this value, you must have supplied a sequence number when you copied the subfile using a `dfcpy` function.

rstaddr
is the file address to which you want to restore the subfile.

seq
is an update sequence number. The number you provide must match the sequence number contained in the subfile that is specified by the *rstaddr* parameter.

If the numbers do not match, the `dfrst` function does not proceed and issues an error return. If the numbers match, the `dfrst` function restores the subfile and increases the sequence number by 1. This sequence number is placed in the prime block of the restored subfile.

Entry Requirements

Before you can use the `dfrst` function, you must make a copy of a subfile using the `dfcpy` function. You can modify this copy using other TPFDF functions.

Normal Return

The restored file (with any modifications you have made to the copy) becomes the currently open subfile. You can continue processing it using other TPFDF functions. The copy is released unless you use the DFRST_FLIP value.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_xxx`. See the description of the specific parameter for information about what type definition to use for that parameter.
- The `dfrst` function rebuilds the B+Tree index for B+Tree files.

Examples

- The following example restores a subfile that might be being held (by this or another application).

```
dft_fad fad;
dft_fil *file_ptr;
   .
   .
   .
dfrst(file_ptr, DFRST_HELD, fad);
```

- The following example restores a subfile from a specified file address (contained in file_address).

```
dft_fil *file_ptr;
dft_fad file_address;
   .
   .
   .
dfrst(file_ptr, 0, file_address);
```

Related Functions

“dfcpy–Copy a Subfile” on page 92.

df_setkey–Setting Up a Key in a Key List

Use this group of functions to set up a key list. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

A key list is activated using the `dfkey` function.

Format

```
void df_setkey(dft_kyl *key_list, short int nbr,
     short int dsp, short int len, char con,
     char *sea, char msk, char org, char typ);
void df_setkey_bool(dft_kyl *key_list, short int nbr,
     short int dsp, short int len, char con,
     char *sea, char msk, char org, char typ,
     char bool);
void df_setkey_dbdef(dft_kyl *key_list, char *sea,
     char msk);
void df_setkey_mod(dft_kyl *key_list, short int nbr,
     short int dsp, short int len, char *sea,
     char msk, char oper);
```

bool
is the type of Boolean connector between the current and subsequent Boolean slots. Use one of the following values:

DF_OR
specifies the OR connector

DF_AND
specifies the AND connector

DF_ORIF
specifies the ORIF connector

DF_ANDIF
specifies the ANDIF connector.

con
is the condition that must exist for the match to be successful.

Use one of the following values if you specify the DF_CONST, DF_CHAR, or DF_PACKED value for the *typ* parameter:

DF_EQ
specifies the LREC key field is equal to the search argument

DF_NE
specifies the LREC key field is not equal to the search argument

DF_GT
specifies the LREC key field is greater than the search argument

DF_LE
specifies the LREC key field is less than or equal to the search argument

DF_LT
specifies the LREC key field is less than the search argument

DF_GE
specifies the LREC key field is greater than or equal to the search argument

Use one of the following values if you specify the DF_MASK value for the *typ* parameter:

DF_Z
specifies the result is all zeros.

DF_O
specifies the result is all ones.

DF_M
specifies the result is mixed ones and zeros.

DF_NZ
specifies the result is not all zeros.

DF_NM
specifies the result is not mixed ones and zeros.

DF_NO
specifies the result is not all ones.

dsp
is the displacement (in bytes) of the key field in the LREC. For example, if the LREC is a variable-length LREC and you want to specify the LREC ID as the key field, the value is 2.

key_list
is a pointer to the key list that will be used to set up the active keys.

len
is the length (in bytes) of the key field in the LREC. For example, if you want to specify the LREC ID as the key field, the value is 1. If you are using a mask field (value DF_MASK of the *typ* parameter), *len* must be 1.

msk
is one of the following base on the function you are using:

- For the `df_setkey`, `df_setkey_bool`, and `df_setkey_mod` functions, *msk* is a 1-byte search argument or a 1-byte mask, for example, 0xFF. Set this to zero when you are using the *sea* parameter.
- For the `df_setkey_dbdef` function, *msk* is the default key LREC ID. For read-only default keys, use X'01'–X'0F'; for read and add operations, use X'10'–X'EF'.

nbr
is the key number you are setting up. Use a number from 1–180 for a selection key list or a sort/merge key list. Use a number from 1–6 for a modification key list. Use 1 for a default-key key list.

oper
specifies the operation to perform during a global modification of LRECs. The operation is applied to the key fields in the LRECs using the values in the modification key list.

Use one of the following values to indicate the operation to be performed on the LRECs being globally modified:

DF_MVI
moves the value contained in SW01MSK into the LREC at the displacement specified by SW01DIS.

DF_MVC
moves the character string whose address is in SW01SEA into the LREC, starting at the displacement specified by SW01DIS for the length contained in SW01LEN.

DF_FILL
propagates the character contained in SW01MSK into the LREC, starting at the displacement specified by SW01DIS for the length contained in SW01LEN.

DF_OI
performs an OR-Immediate (OI) operation on the byte in the LREC at the displacement in SW01DIS by using the value specified in SW01MSK.

DF_OC
performs an OR-Character (OC) operation in the LREC beginning on the byte whose displacement is specified in SW01DIS for a length given in SW01LEN by using the value whose address is given in SW01SEA.

DF_NI
performs an AND-Immediate (NI) operation on the byte in the LREC at the displacement in SW01DIS by using the value specified in SW01MSK.

DF_NC
performs an AND-Character (NC) operation in the LREC beginning on the byte whose displacement is specified in SW01DIS for a length given in SW01LEN by using the value whose address is given in SW01SEA.

DF_XI
performs an Exclusive OR-Immediate (XI) operation on the byte in the LREC at the displacement in SW01DIS by using the value specified in SW01MSK.

DF_XC
performs an Exclusive OR-Character (XC) operation in the LREC beginning on the byte whose displacement is specified in SW01DIS for a length given in SW01LEN by using the value whose address is given in SW01SEA.

DF_ADD
adds the fullword value whose address is in SW01SEA to the fullword value in the LREC whose displacement is specified by SW01DIS.

DF_ADD_SHORT
adds the halfword value whose address is in SW01SEA to the halfword value in the LREC whose displacement is specified by SW01DIS.

DF_SUB
subtracts the fullword value whose address is in SW01SEA to the fullword value in the LREC whose displacement is specified by SW01DIS.

DF_SUB_SHORT
subtracts the halfword value whose address is in SW01SEA to the halfword value in the LREC whose displacement is specified by SW01DIS.

DF_COUNT
increments a fullword counter whose address is in SW01SEA. The application is responsible for initializing the counter before the global modification operation.

DF_COUNT_SHORT
increments a halfword counter whose address is in SW01SEA. The application is responsible for initializing the counter before the global modification operation.

DF_SUM
adds the fullword value in the LREC at the displacement specified in SW01DIS to the fullword sum whose address is in SW01SEA. The sum must be initialized before the global modification operation.

DF_SUM_SHORT
adds the halfword value in the LREC at the displacement specified in SW01DIS to the halfword sum whose address is in SW01SEA. The sum must be initialized before the global modification operation.

DF_MAX
finds the maximum value of the fullword value in the LREC at the displacement specified by SW01DIS and the current fullword maximum whose address is in SW01SEA. The new maximum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation.

DF_MAX_SHORT
finds the maximum value of the halfword value in the LREC at the displacement specified by SW01DIS and the current halfword maximum whose address is in SW01SEA. The new maximum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation.

DF_MIN
finds the minimum value of the fullword value in the LREC at the displacement specified by SW01DIS and the current fullword minimum whose address is in SW01SEA. The new minimum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation.

DF_MIN_SHORT
finds the minimum value of the halfword value in the LREC at the displacement specified by SW01DIS and the current halfword minimum whose address is in SW01SEA. The new minimum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation.

org
specifies the organization of the key fields. Use one of the following values:

DF_DOWNORG
specifies that the subfile is DOWN organized on this key field.

DF_UPORG
specifies that the subfile is UP organized on this key field.

DF_NOORG
specifies that the subfile is not organized on this key field.

sea
is one of the following based on the function you are using:

- For the `df_setkey`, `df_setkey_bool`, and `df_setkey_mod` function, *sea* is the address of the search argument. Set this to zero when you are using the *msk* parameter.

- For the df_setkey_dbdef function, *sea* is a pointer to the prototype logical record (LREC). The LREC ID contained in the prototype LREC must be in the range of nonread-only default keys (X'10' to X'EF').

typ
is the type of search argument you are specifying. Use one of the following values to use the contents of the *msk* parameter for the search:

DF_CONST
specifies that the *msk* parameter contains a 1-byte search argument.

DF_MASK
specifies that the *msk* parameter contains a 1-byte mask.

Use one of the following values to use the contents of the string pointed to by the *sea* parameter for the search:

DF_CHAR
specifies that the search argument is a variable-length character string.

DF_PACKED
specifies that the search argument is a variable-length packed decimal string.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

- The type definitions (for example, dft_fil, dft_ref, and dft_kyl) are defined in the c$cdfapi.h header file.
- If any key is set to DF_NOORG, all subsequent keys must be set to DF_NOORG.
- If you supply a mask (*typ* set to DF_MASK), the organization must be DF_NOORG.
- If you specify a mask, the df_setkey and df_setkey_bool functions perform a bitwise AND (&) operation between the key field in the LREC and the contents of the *msk* parameter.
- If you use the df_setkey_bool function for any key in a key list, you must use it for every key in the key list except for the last. Use the df_setkey function for the last key.
- When Boolean operators are specified in a key list, the file is treated as NOORG. That is, it is treated as a file that has no organization even if it has UP or DOWN organization. This may have an impact on the response time of functions that use the key list, such as the dfred, dfdel, and dfmod functions. Furthermore, if the file has a B^+Tree index file associated with it, the index file will not be used when retrieving LRECs.
- The df_setkey_mod function is used to set up the rules for global modifications but does not select the records to which those rules are applied.

- You can define any number of key list structures in your program. Each key list can have from 1–180 keys. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

Examples

- See “Using a Key List with the df_setkey Function” on page 29 and “df_nbrkeys–Setting Up the Number of Keys” on page 124 for examples of how to use the `df_setkey` function.
- See “dfred–Read a Logical Record” on page 134 for an example of how to use the `df_setkey_bool` and `df_setkey_dbdef` functions.
- See “dfmod–Perform or Indicate Logical Record Modifications” on page 117 for an example of how to use the `df_setkey_mod` function.
- See “Setting up a Key List with Less than Six Keys” on page 29 and “Setting up a Key List in the Range 1-180” on page 30 for examples of how to set up a key list.

Related Functions

- “df_nbrkeys–Setting Up the Number of Keys” on page 124
- “dfkey–Activate a Key List” on page 115.

dfspa–Create Work Space

Use this function to obtain and initialize work space linked to the SW00SR slot for a subfile. This space is available while the subfile is open.

Format

```
void *dfspa(dft_fil *file, dft_spc spc, dft_sps sps);
```

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

spc
is the character you want to use to initialize the work space.

sps
is the size of the space, which can be a maximum of 3952 bytes.

Entry Requirements

None.

Normal Return

The address of the space that the TPFDF product has provided. The TPFDF product also loads this address in the SW00WKA field of the SW00SR slot.

Error Return

None.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- If you allocate work space with the `dfopn` function and you then call the `dfspa` function for the same file, the TPFDF product returns the space originally allocated.
- If you set the *sps* parameter to zero, the TPFDF product releases any space previously allocated by a `dfspa` or `dfopn` function.

Examples

The following example creates a 400-byte area filled with space characters. The TPFDF product puts a pointer to the space in SW00WKA.

```
dft_fil *file_ptr;
   .
   .
   .
dfspa(file_ptr, ' ', 400);
```

Related Functions

“dfopn–Open a Subfile” on page 125.

dfsrt–Sort a Subfile

Use this function to sort logical records (LRECs) in an open subfile.

Format

```
void dfsrt(dft_fil *file, dft_fil *input, dft_opt options, dft_kyl *key_list);
void dfsrt_pty(dft_fil *file, dft_fil *input, dft_opt options,
     dft_kyl *key_list, dft_pty pty);
```

Options Parameter Values:

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

input
is a pointer to the SW00SR slot of the subfile that you want to be sorted.

key_list
is the address of the key list specifying the order into which you want the TPFDF product to sort the LRECs. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

options
are the processing options for this function. Use the following values:

DFSRT_FULLFILE
sorts LRECs from the entire input file (specified in *input*), not from a single subfile.

DFSRT_RELEASE
releases the SW00SR slot of the input subfile (specified in *input*).

DFSRT_RELFC
releases the input subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

0 specifies that you do not want to use any processing options.

pty
is the pool type of the overflow blocks, which can be one of the following:

0 uses the pool type defined by the PF0 parameter of the DBDEF macro.

1 uses the pool type defined by the PF1 parameter of the DBDEF macro.

2 uses the pool type defined by the PF2 parameter of the DBDEF macro.

Entry Requirements

Both subfiles must be opened before you call the `dfsrt` function.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent functions when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with functions that access or update records without using fullfile processing.
- Any keys that are active when you call this function are used to select records from the input file.
- The keys that you specify with this function are used to sort the LRECs in the output subfile unless the file is a B+Tree file. The *keylist* parameter is ignored for B+Tree files. The output file is organized according default keys defined on the DBDEF macro for the file. See *TPFDF Database Administration* for more information about default keys.
- The `dfsrt` function does not change the input subfile.
- After processing, the original contents of the output subfile are lost.
- When the `dfsrt` function completes processing, the output subfile is left open and must be closed using the `dfcls` function before the ECB exits. If you specify the DFSRT_RELEASE value, the `dfsrt` function closes the input file.
- If you sort a large file in detac mode, the TPFDF product puts the sorted output in a newly created pool file and closes the output file that was originally specified. You must restore the file to a fixed file to permanently save the results.
- You cannot issue additional TPFDF functions to the input file until the file is closed if the following conditions are true:
  - You specify the DFSRT_FULLFILE option
  - You do not specify the DFSRT_RELEASE option
  - The end-of-file indicator is set.

  However, you can specify the DFCLS_REUSE option on the `dfcls` function. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- You cannot use this function with P-type files, add current files, or pushdown chaining files.
- Figure 19 shows how the `dfsrt` function sorts LRECs from one subfile into another.

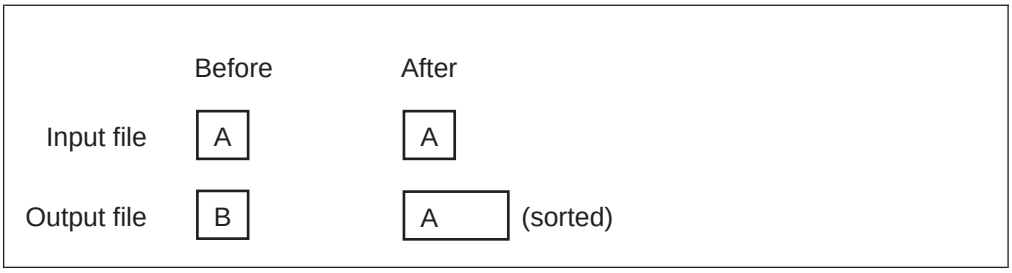


Figure 19. Sorting LRECs from One Subfile into Another. The input file is defined by the input parameter and the output file is defined by the file parameter.

- If you use the `dfsrt` function in a commit scope, both input and output files must be opened in the same commit scope. See "Commit Scopes" on page 8 for more information about commit scopes.
- When you use the `dfsrt` function to sort large input files, use detac mode for the output files to ensure optimum system performance.

Examples

The following example sorts LRECs from the subfile file_ptr and puts them into the file out_file_ptr. The sort key is in a key list structure called keys.

```
/* set up the keys to use to sort the output file */

df_nbrkeys(&sortkeys, 1);

df_setkey(&sortkeys, 1, offsetof(struct gr95sr, gr95nam),
          member_size(struct gr95sr, gr95nam),
          0, NULL, 0, DF_UPORG, DF_CHAR);

/* set up the keys to use to select the LRECs in the input file */

df_setkey(&keys, 1, offsetof(struct gr95sr, gr95key),
          1, DF_EQ, &pky, 0, DF_UPORG, DF_CHAR);


df_nbrkeys(&keys, 1);

dfkey(in_fileptr, &keys);

/* sort the subfile after extracting matching LRECs            */
/* release the input file after the sort                       */
/* (the key list in the command is the sort key specification) */

dfsrt(out_file_ptr, in_fileptr, DFSRT_RELEASE, &sortkeys);
```

Related Functions

"dfmrg–Merge Logical Records from Two Subfiles" on page 121.

dftld–Write a Subfile from Main Storage to DASD

Use this function to do one of the following:

- Write the subfile to DASD
- Ignore the subfile.

Format

```
void dftld(dft_fil *file, dft_opt options);
void dftld_acc(dft_fil *file, dft_opt access, dft_opt options,
     dft_xxx acc);
```

Access Parameter Values:

Options Parameter Values:

acc
: is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
: is the method you want to use to access the subfile. Use one of the following values:

  DFTLD_ALG
  : specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

    If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFTLD_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFTLD_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

options
are the processing options for this function. Use the following values:

DFTLD_CREATE
writes the subfile to new pool blocks in DASD. (The default is to use the same file addresses as before.)

DFTLD_SKIP
discards the blocks that were read from tape or sequential data set with the `dftrd` function. The `dftld` function releases all the blocks, both prime and chained, that the `dftrd` function retrieved and placed in main storage.

You can use the DFTLD_SKIP value in a restart situation when a number of blocks need to be read from tape or sequential data set to reach the point where a system failure occurred. (All the blocks up to the failure point have already been written to DASD, so you only need to read them without saving them again.)

The DFTLD_SKIP value is also useful if you want to end the transfer of information from tape to disk, or if there are unwanted blocks of data on a tape or sequential data set.

0 specifies that you do not want to use any processing options.

Entry Requirements

You must successfully read a subfile using the `dftrd` function before calling the `dftld` function.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these

parameters is shown as dft_*xxx*. See the description of the specific parameter for information about what type definition to use for that parameter.

- You must always call dftld after calling dftrd. No other function calls are allowed between the dftld and dftrd calls. Ensure your application checks for any error conditions before calling dftld. During dftrd processing, if an error occurs that causes SW00RTN to be nonzero, invoking dftld is treated as an incorrect command sequence.
- The dftld function rebuilds the B+Tree index for B+Tree files.
- Because the dftld function requires a significant amount of system resources, do not use this function in a commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

The following example writes a subfile, specified in alg_ptr, to new pool blocks in DASD. It ignores any blocks that were read from tape by a dftrd.

```
dft_fil *file_ptr;
dft_alg *alg_ptr;
   .
   .
   .
dftld_acc(file_ptr, DFTLD_ALG, DFTLD_SKIP | DFTLD_CREATE, alg_ptr);
```

Related Functions

- “dftrd–Read a Subfile from an Input Tape to Main Storage” on page 166
- “dftlg–Write a File or Subfile to Tape” on page 163.

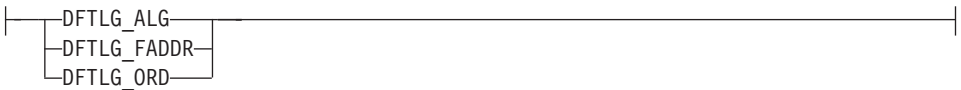
dftlg–Write a File or Subfile to Tape

Use this group of functions to write a file, or part of a file, to a real-time tape, a general tape, or a sequential data set.

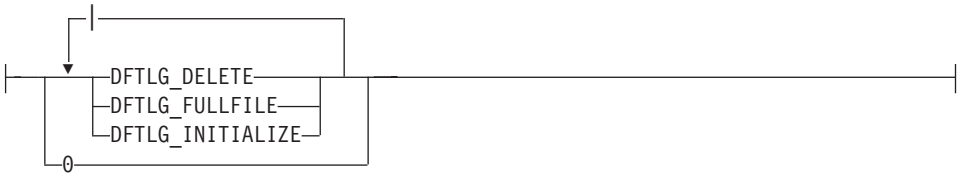
Format

```
void dftlg(dft_fil *file, dft_opt options, dft_tpn *tape);

void dftlg_acc(dft_fil *file, dft_opt access, dft_opt options,
     dft_tpn tape, dft_xxx acc);

void dftlg_inc(dft_fil *file, dft_opt options, dft_tpn *tape,
     dft_idl *inc_list);

void dftlg_acc_inc(dft_fil *file, dft_opt access, dft_opt options,
     dft_tpn tape, dft_xxx acc, dft_idl *inc_list);
```

Access Parameter Values:



Options Parameter Values:



acc
is an ordinal number, a file address, or a pointer to an algorithm string that specifies the subfile you want to access. The type for this parameter is determined by the value you specify for the *access* parameter.

access
is the method you want to use to access the subfile. Use one of the following values:

DFTLG_ALG
specifies that you are providing a pointer to an algorithm argument in the *acc* parameter. The *acc* parameter is of type `dft_alg`. The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database*

Administration for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

DFTLG_FADDR
specifies that you are providing a file address in the *acc* parameter. A file address is in integer format. The *acc* parameter is of type `dft_fad`.

DFTLG_ORD
specifies that you are providing an ordinal number in the *acc* parameter. Ordinal numbers in a file start at zero and are in integer format. The *acc* parameter is of type `dft_ord`.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

inc_list
is a pointer to a list of file IDs that you want to be included in the action of the function.

options
are the processing options for this function. Use the following values:

DFTLG_DELETE
deletes all the LRECs in the DASD file after the records are written to tape or sequential data set. Any previously used blocks are returned to pool.

DFTLG_FULLFILE
writes LRECs from the whole input file (not from a single subfile) to tape or sequential data set.

DFTLG_INITIALIZE
deletes all the LRECs in the DASD file after the records are written to tape or sequential data set. Any previously used blocks are returned to pool, except the prime block.

0 specifies that you do not want to use any processing options.

tape
is a pointer to a character array holding the 3-character tape name.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.

- Some parameters can be of different types based on the value you specify for that parameter or a related parameter. In the function format, the type of these parameters is shown as `dft_xxx`. See the description of the specific parameter for information about what type definition to use for that parameter.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent functions when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with functions that access or update records without using fullfile processing.
- The `dftlg` function does not write the B+Tree index information to tape because it is rebuilt when the B+Tree data file is restored.
- If you specify the DFTLG_FULLFILE value and the end-of-file indicator is set, you cannot issue additional TPFDF functions until the file is closed. However, you can specify the DFCLS_REUSE option on the `dfcls` function. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- If you use the `dftlg` function in a commit scope, a rollback of the commit scope will not restore the contents or the position of the tape. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

- The following example writes a complete file to tape. A pointer to the 3-character tape name is in tape_name.

```
dft_tpn *tape_name;
dft_fil *file_ptr;
   .
   .
   .
dftlg(file_ptr, 0, tape_name);
```

- The following example writes a subfile, specified in alg_ptr to tape. A pointer to the tape name is in tape_name. It deletes LRECs in the subfile after the write.

```
dft_tpn *tape_name;
dft_alg *alg_ptr;
dft_fil *file_ptr;
   .
   .
   .
dftlg(file_ptr, DFTLD_ALG, DFTLG_INITIALIZE, tape_name, alg_ptr);
```

Related Functions

- “dftld–Write a Subfile from Main Storage to DASD” on page 160
- “dftrd–Read a Subfile from an Input Tape to Main Storage” on page 166.

dftrd–Read a Subfile from an Input Tape to Main Storage

Use this function to read a subfile from an input tape or sequential data set to main storage.

Format

```
void dftrd(dft_fil *file, dft_tpn *tape);
```

file
is a pointer to the base address of the SW00SR slot (defined in `c$sw00sr.h`) of the subfile that you want to access and is returned by the `dfopn` function.

tape
is a pointer to a variable containing the 3-character symbolic tape name or sequential data set.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The type definitions (for example, `dft_fil`, `dft_ref`, and `dft_kyl`) are defined in the `c$cdfapi.h` header file.
- The `dftrd` function reads one subfile at a time. After you call the `dftrd` function, you must then call the `dftld` function before reading another subfile. (You can use the DFTLD_SKIP value with `dftld` to ignore the subfile.)
- If you use the `dftrd` function in a commit scope, a rollback of the commit scope will not restore the position of the tape. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

The following example reads a subfile from tape to main storage. A pointer to the 3-character tape name is in tape_name.

```
dft_tpn *tape_name;
dft_fil *file_ptr;
   :
   :
dftrd(file_ptr, tape_name);
```

Related Functions

- “dftld–Write a Subfile from Main Storage to DASD” on page 160
- “dftlg–Write a File or Subfile to Tape” on page 163.

dfuky–Generate a Unique Key for Use in Logical Records

Use this function to generate a unique key in the SW00SR slot for the subfile.

Format

```
char *dfuky(dft_fil *file);
```

file
is a pointer to the base address of the SW00SR slot (defined in c$sw00sr.h) of the subfile that you want to access and is returned by the dfopn function.

Entry Requirements

None.

Normal Return

A pointer to the 4-byte unique key value returned by the TPFDF product.

Error Return

- See "Identifying Return Indicators and Errors" on page 13 for information about how to check the error indicators.
- A zero value in the SW00UKY field of the SW00SR indicates an error.

Programming Considerations

- The type definitions (for example, dft_fil, dft_ref, and dft_kyl) are defined in the c$cdfapi.h header file.
- The value that is returned in the SW00UKY field of the SW00SR is only valid immediately after the dfuky function is processed. Subsequent TPFDF functions reuse and overwrite this field.
- In order to use the unique key feature, the file header must be expanded by 18 bytes.
- See "Grouping LRECs Together Using the Unique Key Facility" on page 6 for more information about using unique keys.

Examples

The following example generates a unique key value for the file. The value is placed in the SW00UKY field in the SW00SR slot.

```
dft_fil *file_ptr;
   .
   .
   .
dfuky(file_ptr);
```

The following is an example of a header expanded by 18 bytes so that the unique key feature can be used.

```
IRXXHDR& DS  CL16         STANDARD FILE HEADER
         DS  CL10         STANDARD TPFDF HEADER
         DS  CL18         UNIQUE KEY HEADER EXTENSION
```

Related Functions

"dfkey–Activate a Key List" on page 115.

member_size–Calculating the Size of a Structure Member

Use this function to calculate the size of any member of a structure.

Format

```
unsigned int member_size(s, m);
```

m is the member name.

s is the structure or union name.

Entry Requirements

None.

Normal Return

The size (in bytes) of the member.

Error Return

None.

Programming Considerations

None.

Examples

The following example defines various numbers:

```
#define MEM_NUM_SIZE    member_size(struct ir00df,lrec.rnum.ir00num)
#define MAX_NAME_SIZE   member_size(struct ir00df,lrec.rnum.ir00nam)
#define PMNT_SIZE       member_size(struct ir00df,lrec.rnum.ir00pay)
```

Related Functions

None.

TPFDF Restricted C Language Functions: Reference

This chapter documents Product-Sensitive Programming Interface information.

The functions in this section are TPFDF internal functions used to perform system functions. There is no guarantee that the programming interface to these functions will not change. The use of these functions should be restricted to minimize the effect of any changes.

This chapter contains an alphabetic listing of the TPFDF restricted C language functions. The description of each function includes the following information:

Format: Provides the function prototype and a description of each parameter and variable.

Entry Requirements: Lists any special conditions that must be true when you use the function.

Normal Return: Lists what is returned when the function has completed processing successfully.

Error Return: Lists what is returned when the function cannot complete processing successfully.

Programming Considerations: Lists any additional considerations for using the function, including any restrictions or limitations.

Examples: Provides one or more examples that show you how to code the function.

Related Functions: Lists where to find information about related functions.

© Copyright IBM Corp. 1997, 2001

dftab–Access Database Definition Tables

Use the following functions to access the database definition (DBDEF) tables.

Format

```
SW02SR_PTR dftab(dft_opt opt,
           dft_fid *id_ptr, dft_fvn fvn,
           dft_rct rct, SW02SR_PTR dfssu_ptr);
SW02SR_PTR dftab_id(dft_fid *id_ptr,
           dft_fvn fvn, SW02SR_PTR dfssu_ptr);
SW02SR_PTR dftab_rct(dft_rct rct,
           SW02SR_PTR dfssu_ptr);
SW02SR_PTR dftab_base();
```

opt
is the processing option. Use one of the following values:

DFTAB_ID
locates the database definition (DBDEF) table for the specified file identifier (ID). If you specify this value, specify 0 for the *rct* parameter.

DFTAB_RCT
locates the DBDEF table for the specified record type. Record types are numeric values defined in the TPF SYSEQC macro or ALCS DXCURID macro. If you specify this value, specify NULL for the *id_ptr* parameter and 0 for the *fvn* parameter.

DFTAB_BASE
locates the base of the DBDEF index table. If you specify this value, specify NULL for the *id_ptr* parameter and 0 for the *fvn* and *rct* parameters.

id_ptr
is a pointer to a 2-byte field containing the file ID.

fvn
is the file version number.

rct is the record type.

dfssu_ptr
is a pointer to a work area for subsystem user (SSU) information. This applies only to a multiple database function (MDBF) environment in a TPF system. The work area must be at least 381 bytes.

If you do not have an MDBF environment, specify NULL for this parameter.

Entry Requirements

None.

Normal Return

- If you specify the `dftab_id` function or the DFTAB_ID value, a pointer to the DBDEF table for this file ID and file version number is returned.
- If you specify the `dftab_rct` function or the DFTAB_RCT value a pointer to the DBDEF table for this record type is returned.
- If you specify the `dftab_base` function or the DFTAB_BASE value a pointer to the DBDEF index table is returned.

Error Return

NULL.

Programming Considerations

You can use the `dftab` function with the appropriate value specified for the *opt* parameter or you can use the `dftab_id`, `dftab_rct`, or `dftab_base` function to accomplish the same task.

Examples

The following code checks the existence of database definitions with a file ID of AB in a non-MDBF environment.

```
#include <c$cdf.h>

if (dftab_id("AB",0,NULL) == NULL) {
  wtopc_text("ID not defined in DBDEF");
}
```

Related Functions

None.

Part 3. Assembler Macros

TPFDF General-Use Assembler Macros: Reference 175
DBADD–Add a Logical Record to a Subfile 176
DBADR–Provide the File Address of a Prime Block 190
DBCKP–Checkpoint a Subfile 195
DBCLR–Allow ECB Exit with Open Files 198
DBCLS–Close a Subfile 199
DBCPY–Copy a Subfile 205
DBCRE–Create a Subfile 210
DBDEL–Delete One or More Logical Records 214
DBDIX–Delete Index References to a Subfile 228
DBDSP–Display Logical Records from a Subfile 231
DBFRL–Ensure an ECB Data Level Is Free 240
DBIDX–Create an Index Reference 241
DBIFB–Check a SW00SR Slot 244
DBKEY–Activate a Key List 247
DBMOD–Perform or Indicate Logical Record Modifications 249
DBMRG–Merge Logical Records from Two Subfiles 254
DBOPN–Open a Subfile 260
DBRED–Read a Logical Record 272
DBREP–Replace a Logical Record with Another Logical Record 286
DBRET–Retain a Logical Record Position 290
DBRST–Restore a Subfile 293
DBSETK–Setting Up a Key in a Key List 298
DBSPA–Create Work Space 304
DBSRT–Sort a Subfile 307
DBTLD–Write a Subfile from Main Storage to DASD 313
DBTLG–Write a File or Subfile to Tape 318
DBTRD–Read a Subfile from an Input Tape to Main Storage 323
DBUKY–Generate a Unique Key for Use in Logical Records 325

TPFDF Restricted Assembler Macros: Reference 327
BLKSZ–Convert a Block Type to a Block Size 328
DBCNT–Calculate the Length of an Assembler Symbol 332
DBTAB–Access Database Definition (DBDEF) Tables 333
DFCAS–TPFDF Case Setup in Fast-link Segments 336
DFCLIB–C Language Interface 338
DFDDA–Distributed Data Access Support 341
DFDLAY–Delay Processing Conditionally 342
DFGDS–General Data Set Support User Exit 343
DFGETC–Get Working Storage Block 344
DFGLVL–Get Resource Level 346
DFGPNL–Get Calling Program Address 347
DFLNK–TPFDF Fast Linkage 348
DFSSU–Handling DBDEF Subtables 350
DFTDC–Dialogue Control Facility Support User Exit 353
DFUEX–Define TPFDF User Exit Point 354
FILTP–Determine File Address Type 355
FMSGS–Set Up Output Messages 357
HELPA–Help Message Text 360

© Copyright IBM Corp. 1997, 2001

TPFDF General-Use Assembler Macros: Reference

This chapter contains an alphabetic listing of the TPFDF assembler macros that you can use in application programs. The description of each macro includes the following information:

Format: Provides a syntax (railroad track) diagram for the macro and a description of each parameter and variable. See “How to Read the Syntax Diagrams” on page xiv for more information about syntax diagrams.

Entry Requirements: Lists any special conditions that must be true when you use the macro.

Normal Return: Lists what is returned when the macro has completed processing successfully.

Error Return: Lists what is returned when the macro cannot complete processing successfully.

Programming Considerations: Lists any additional considerations for using the macro, including any restrictions or limitations.

Examples: Provides one or more examples that show you how to code the macro.

Related Macros: Lists where to find information about related macros.

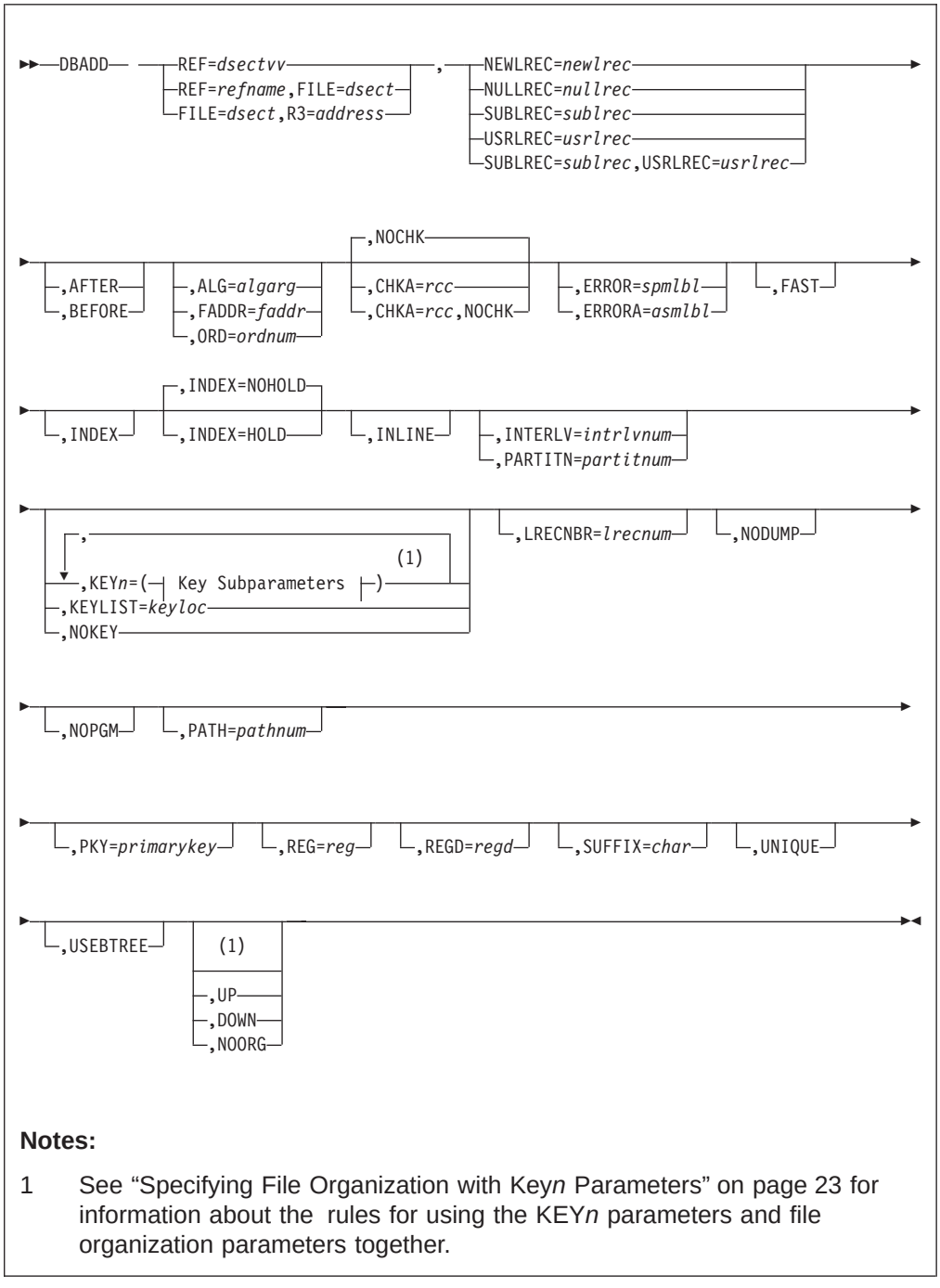
© Copyright IBM Corp. 1997, 2001

DBADD–Add a Logical Record to a Subfile

Use this macro to do the following:

- Add a fixed- or variable-length logical record (LREC) to a subfile
- Add an empty LREC to a subfile
- Add a subLREC to the current extended LREC
- Add an extended LREC to a subfile
- Add an extended LREC and a subLREC to a subfile.

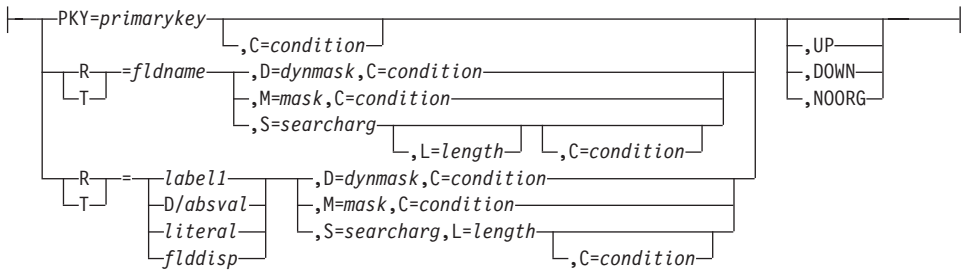
Format



Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

Key Subparameters:



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

NEWLREC=*newlrec*
adds a new fixed-length or variable-length LREC, where *newlrec* is one of the following:

- A register that contains the address of the LREC to add
- A label in one of the following formats:

  newlrec
  is the label of a field that contains the LREC.

  AI*newlrec*
  is the label of a 4-byte field that contains the storage location of the LREC.

NULLREC=*nullrec*
adds an empty LREC to a subfile, where *nullrec* is one of the following:

- A register that contains the address of a 2-byte field that contains the length of the LREC

- A label in one of the following formats:

  nullrec
  is the label of a 2-byte field that contains the length of the LREC.

  AI*nullrec*
  is the label of a 4-byte field that contains the storage location of the 2-byte length of the LREC.

If you specify this parameter, you must also specify the primary key (LREC ID) of the empty LREC with the PKY parameter.

Notes:

1. Do not use the NULLREC parameter on files that are UP or DOWN organized because it can destroy the organization of the file.
2. NULLREC is most suitable for W-type files.
3. NULLREC is not allowed for B^+Tree files.

SUBLREC=*sublrec*
adds a subLREC to the current extended LREC or to the extended LREC specified by the USRLREC parameter, where *sublrec* is one of the following:

- A register that contains the address of the subLREC
- A label in one of the following formats:

  sublrec
  is the label of a field that contains the subLREC.

  AI*sublrec*
  is the label of a 4-byte field that contains the storage location of the subLREC.

  If you specify this parameter for an LREC that contains existing subLRECs, the subLREC is added immediately before the existing subLRECs. See “Using Extended Logical Records” on page 4 for more information about how subLRECs are added to an extended LREC.

USRLREC=*usrlrec*
adds an extended LREC to the subfile, where *usrlrec* is one of the following:

- A register that contains the address of the userLREC
- A label in one of the following formats:

  usrlrec
  is the label of a field that contains the userLREC.

  AI*usrlrec*
  is the label of a 4-byte field that contains the storage location of the userLREC.

AFTER
adds the new LREC immediately after the current LREC.

BEFORE
adds the new LREC immediately before the current LREC.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based

on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FAST
used for migration purposes only; use the INLINE or NOKEY parameter instead. If you specify this parameter, the NOKEY parameter is implemented; that is, any currently active keys are deactivated.

INDEX
adds an LREC to a detail subfile or intermediate index subfile where the index structure does not yet exist. If you specify this parameter, the algorithm defined for the new subfile must be #TPFDBFF.

When you specify this parameter, the subfile is created and indexed by adding an index LREC in the index file referencing the subfile.

INDEX=HOLD
potentially holds any index files that reference the subfiles you are accessing and prevents two or more application programs from modifying the index files at the same time. Holding occurs if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the index file will not occur until the index file is no longer held. If more than one application can update the same index file, you must specify this parameter to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, affect hold processing.

INDEX=NOHOLD
does not hold the index files that reference the subfiles you are accessing.

INLINE
provides inline processing for this macro. You cannot use this parameter with key parameters or with extended LRECs. Any keys that are active from previous TPFDF macros are deactivated; that is, previous key arguments are set to zero.

INTERLV=*intrlvnum*
specifies the number of the interleave that you want to use, where *interlvnum* is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN=*partitnum*
specifies the number of the partition that you want to use, where *partitnum* is one of the following:
- A register that contains the address of the partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm calculates the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

KEY*n*
specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See “Specifying File Organization with Key*n* Parameters” on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

PKY=*primarykey*
specifies a value that will be compared against the primary key of an LREC, where *primarykey* is a 1-byte immediate value; for example:

```
... KEY1=(PKY=#RR00K80)
```

This has the same effect as:

```
... KEY1=(R=RR00KEY,S=#RR00K80)
```

R specifies a field in the LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

T specifies a field in the subLREC of an extended LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,S=EBW000)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=EBX010,S=EBW000,L==H'4')
```

D*l**absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/2,S=EBW000,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/GR00NAM-GR00REC,S=EBW000,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R==H'2',S=EBW000,L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+2,S=EBW000,L==H'4')
```

or

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+L'GR00FLD,S=EBW000,L==H'4')
```

C=*condition*
specifies the condition to be used when comparing fields in the logical record (specified with the R subparameter) with the search argument (specified with the S or PKY subparameter) or with the bit mask (specified with the M or D subparameter).

If you specify the S or PKY subparameter, use one of the following values:

Value	Condition
EQ	Equal (this is the default)
E	Equal
NE	Not equal
GE	Greater than or equal
LE	Less than or equal
GT	Greater than
LT	Less than
H	High
L	Low
NH	Not high
NL	Not low.

If you specify the M or D subparameter, use one of the following values:

Value	Condition
Z	Zeros
O	Ones
M	Mixed
NZ	Not zeros
NO	Not ones
NM	Not mixed.

D=*dynmask*
specifies the label of a 1-byte field containing a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,D=EBW000,C=Z)
```

M=*mask*
specifies a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,M=X'80',C=Z)
```

S=*searcharg*
specifies the search argument to be compared with the LREC field specified with the R or T subparameter, where *searcharg* is one of the following:

- A register that contains the address of the search argument
- A literal that represents the search argument
- A label in one of the following formats:

searcharg
is the label of the search argument.

A*l**searcharg*
is the label of a 4-byte field that contains the storage address of the search argument.

P*l**searcharg*
is the label of a field that contains the search argument in packed decimal format.

If you specify P/*searcharg* or a literal in the form of =P"...", the LREC field and search argument are compared as decimal numbers in packed format. Otherwise, the LREC field and search argument are compared as character data.

L=*length*
specifies the length of the search argument, where *length* is one of the following:

- The address of a 2-byte field containing the length of the search argument
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

NOORG
specifies that the key field is in no particular order in the subfile.

KEYLIST=*keyloc*
specifies a key list that you want to use with this macro, where *keyloc* is one of the following:

- A register that contains the address of the key list
- A label in one of the following formats:

  keyloc
  is a label indicating the address of the key list.

  A*l**keyloc*
  is the label of a 4-byte field that contains the storage address of the key list.

See “Setting Up and Using a Key List” on page 26 for information about how to set up a key list.

NOKEY
deactivates any currently active keys.

LRECNBR=*lrecnum*
specifies the sequence number of an LREC that you want to add, where *lrecnum* is one of the following:

- A register that contains the address of the LREC number
- An immediate value that represents the LREC number

- A label in one of the following formats:

  lrecnum
  is the label of a 4-byte field that contains the LREC number.

  A*lrecnum*
  is the label of a 4-byte field that contains the storage address containing the LREC number.

Notes:

1. Do not use this parameter with files for which default keys are defined.
2. LRECs are numbered in increasing order from the start of the subfile (the first LREC in the prime block has sequence number 1).
3. If you specify this parameter with active keys, only those LRECs that match the key conditions are included in the sequence numbering; LRECs that do not match are ignored.
4. When you specify this parameter, the LREC is added immediately ***after*** the specified LREC.
5. If you specify this parameter for an LREC number that does not exist, the LREC is not added.

NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this macro:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the NODUMP parameter is not recommended because it can prevent system errors from being issued that indicate a critical problem.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH=*pathnum*
specifies the path number for a detail subfile using index support, where *pathnum* is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator. If there is only one index path, do not specify this parameter.

See *TPFDF Database Administration* for more information about path numbers.

PKY=*primarykey*
specifies the primary key (LREC ID) of the LREC that you are adding, where *primarykey* is one of the following:

- An equate that represents the primary key (for example, PKY=#GR00K80)
- An explicit term that represents the primary key (for example X'80')
- A label in one of the following formats:

  primarykey
  is the label of a field that contains the primary key (for example, PKY=EBW000)

AI*primarykey*
is the label of a 4-byte field that contains the storage address of the primary key.

You ***must*** specify this parameter if you are:

- Adding an empty LREC using the NULLREC parameter
- Adding an extended LREC using the USRLREC parameter. This primary key is placed in the *zzzz*KEY field in the control area of the extended LREC.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

UNIQUE
specifies that the TPFDF product should not add the LREC to the subfile if an LREC exists with the same key fields.

Notes:

1. If UNIQUE=YES was specified in the DBDEF macro for a file, all DBADD statements for that file default to UNIQUE.
2. If you specify this option for a file that is not organized (that is, defined as NOORG), you must do one of the following:
   - Define default keys for the file being updated
   - Specify an organization using the UP or DOWN parameter
   - Ensure the KEY*n* or KEYLIST parameter was specified with a previous macro; if not, specify the KEY*n* or KEYLIST parameter with this macro call.

   If you do not have default keys defined or have active keys when adding a unique LREC to a file that is not organized, the TPFDF product issues a system error. Symbol &DB013E in the DBLCL macro controls whether the TPFDF product returns control to the application program or exits the entry control block (ECB) after issuing the error. If you set &DB013E to 0, which is the default setting, the ECB exits. If you set &DB013E to 1, control is returned to the application program. See *TPFDF Installation and Customization* for more information about the DBLCL macro.

USEBTREE
specifies that the B^+Tree index is used when adding an LREC to a subfile. You can use this parameter only on a B^+Tree file. Otherwise, this parameter is ignored by the TPFDF product.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

NOORG
specifies that the LRECs are organized in the subfile in no particular order. (NOORG is the default if subfile organization has not been defined in the DBDEF).

Entry Requirements

- Before you use the AFTER or BEFORE parameter, you must first establish a *current LREC* (for example, using the DBRED macro). You can then specify whether you want to add the new LREC before or after this current LREC by using the BEFORE or AFTER parameter with the DBADD macro. You can add any number of LRECs at this point in a subfile without having to reestablish the current LREC. The last LREC added becomes the current LREC.
- Before using the SUBLREC parameter without the USRLREC parameter, you must first establish the extended LREC to which you want to add the subLREC as the current LREC.

Normal Return

The address of the new LREC is placed in the SW00REC field of the SW00SR slot. If you specify the REG parameter, the address of the new LREC is placed in the specified register and SW00REC.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.
- If there are default keys defined in the database definition (DBDEF) and you use the DBADD macro with an LREC ID that has not been defined as a default key, a system error is issued.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- You cannot specify six KEY*n* parameters (KEY1–KEY6) when both of the following conditions are true:
  - The S subparameter is used on all six keys
  - The length of all keys is greater than 1 (either with an explicit length using the L subparameter or an implied length from the field used in the R or T subparameter).

  If you need this type of key definition, you must use a key list.
- Any active keys are ignored when you use the #TPFDB0D algorithm.
- The following rules determine the value of the record code check (RCC) value used when the TPFDF product creates a new subfile:

  - If you do not specify the NOCHK parameter, the TPFDF product creates new subfiles with a random RCC value.
  - If you specify the NOCHK parameter without the CHKA parameter, the TPFDF product creates new subfiles without an RCC value.
  - If you specify both the NOCHK and CHKA parameters, the TPFDF product creates new subfiles with the RCC value specified with the CHKA parameter.
- Do not use the following parameters for B+Tree files:
  - AFTER
  - BEFORE
  - FAST
  - KEY*n*
  - KEYLIST
  - LRECNBR
  - NULLREC
- Do not use the FAST parameter with extended LRECs. That is, do not specify this parameter with the SUBLREC or USRLREC parameters.
- Do not use the following parameters with the SUBLREC parameter:
  - AFTER
  - BEFORE
  - UNIQUE
- If a file is UP or DOWN organized and does not have default keys defined in the DBDEF macro, include the KEY*n* or KEYLIST parameters to preserve the organization of the LRECs.
- If a file has default keys defined in the DBDEF, do not specify the KEY*n* or KEYLIST parameters with the DBADD macro; the TPFDF product inserts the LREC in the correct place in the subfile. Also, any keys that are active from a previous TPFDF macro will no longer be active after the DBADD macro call.
- When DBADD is coded for an add current file, the following considerations apply:
  - KEY*n* and KEYLIST parameters cannot be specified on the DBADD macro.
  - If keys are active, they will not be used to determine the location of the record being added to the subfile. However, the keys will remain active for any subsequent macros.

  See *TPFDF Database Administration* for more information about add current files.
- To add an LREC when no subfile is defined, the TPFDF product obtains a prime block from pool and inserts the LREC into it. It puts the address of this prime block in the SW00FAD field of the SW00SR slot and the record code check into the SW00WCC field.
- If adding an LREC to a subfile block causes the block to overflow, the TPFDF product gets a new block and chains it to the old one.

  Note: How LRECs are added to a subfile depends on the following factors:
  - For *add current files*, the LREC is added to the end of the subfile. See *TPFDF Database Administration* for more information about add current files.
  - For *pushdown chaining* files, the LREC is added as the last LREC in the prime block of the subfile. See *TPFDF Database Administration* for more information about pushdown chaining files.
  - For P-type files, a new block is added after the current block or at the end of the subfile.
  - If you specify the AFTER or BEFORE parameter, the LREC is added after or before the current LREC.

  - If you have active keys (which can include default keys coded on the DBDEF macro for the file), the LREC is added to the subfile at the specified location.
  - In all other cases, the LREC is added at the end of the subfile.
- You can use the NEWLREC parameter with a P-type file to specify the data contents of the new block.
- You can use the NULLREC parameter with a P-type file to add an empty block, chained to the current block. You can then add data to this block using the DBMOD macro.
- Use the DBADD macro with the NULLREC parameter specified to create a work area for a T-type LREC in the underlying W-type file. Use the DBDEL macro to delete the T-type LREC from the underlying W-type file before exiting the application program.
- If you use the #TPFDB0D algorithm, you must specify one of the following parameters:
  - AFTER
  - BEFORE
  - LRECNBR

Examples

- The following example shows how to store the ECB work areas in two T-type files, RTEWSR and RTEXSR respectively. This frees the work areas for other uses.

  Each LREC contains 115 bytes made up of:
  - Size field (2 bytes)
  - LREC ID (1 byte)
  - File reference (8 bytes)
  - Data from work area (104 bytes).

  The files are set up by coding:

```
DBADD REF=RTEWSR,REG=R4,NULLREC==AL2(115)
MVC 11(104,R4),EBW000
DBADD REF=RTEXSR,REG=R4,NULLREC==AL2(115)
MVC 11(104,R4),EBX000
```

- The following example shows how you can add an LREC to a subfile using an algorithm argument. In this example, EBW001 contains the algorithm argument and EBX000 points to the LREC that will be added.

```
DBADD REF=GR45DF,ALG=EBW001,NEWLREC=EBX000
```

- The following example adds an empty LREC that has the length of field GR23L80. The primary key is the value defined for #GR23K80.

```
DBADD REF=GR23DF,NULLREC==AL2(L'GR23L80),PKY=#GR23K80
```

- The following example adds a subLREC and a userLREC.

```
DBADD REF=GR39DF,USRLREC=GR39REC,SUBLREC=A/EBW000,PKY=#GR39K80
```

- The following example shows the use of the SUFFIX parameter.

```
GR25DF REG=R6,SUFFIX=X
DBOPN REF=GR25DF,REG=R6,SUFFIX=X
DBADD REF=GR25DF,SUFFIX=X,                *
      KEY1=(PKY=#GR25K80,UP),             *
      KEY2=(R=GR25ALCX,S=GR25ALC,DOWN)
```

- The following example adds an LREC to a detail subfile.

```
DBADD REF=GR23DF,INDEX,ALG=EBW044
```

  This has the same effect as the following sequence of macros:

```
DBCRE REF=GR23DF
DBIDX REF=GR23DF,ALG=EBW0044
DBADD REF=GR23DF
```

Related Macros

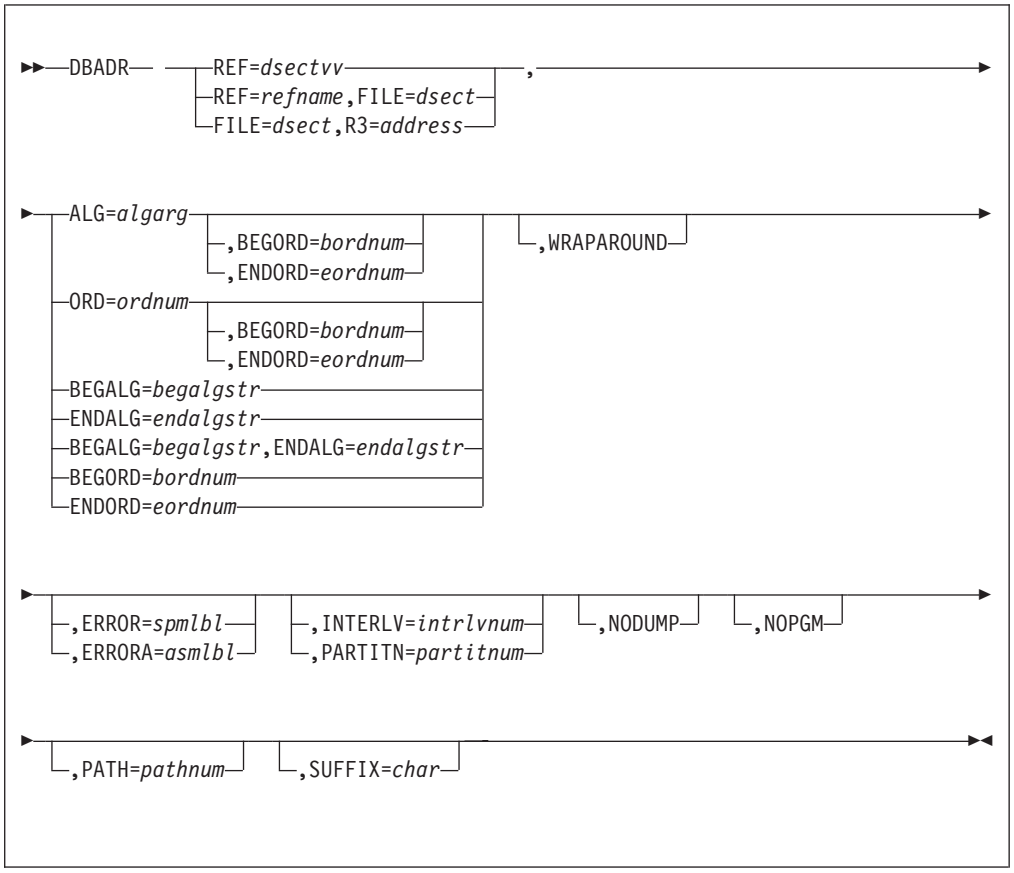
- “DBDEL–Delete One or More Logical Records” on page 214
- “DBMOD–Perform or Indicate Logical Record Modifications” on page 249
- “DBOPN–Open a Subfile” on page 260
- “DBRED–Read a Logical Record” on page 272.

DBADR–Provide the File Address of a Prime Block

Use this macro to get the file address of a prime block in a fixed file.

You can also use the DBADR macro to specify a range of ordinals to be used in subsequent fullfile processing.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

BEGALG=*begalgstr*
calculates the begin ordinal number for use in a subsequent macro that specifies the FULLFILE parameter, where *begalgstr* is one of the following:

begalgstr
is the label of a field that contains the algorithm string used to calculate the ordinal number.

A*l**begalgstr*
is the label of a 4-byte field that contains the storage address of the algorithm string used to calculate the ordinal number.

ENDALG=*endalgstr*
calculates the end ordinal number for use in a subsequent macro that specifies the FULLFILE parameter, where *endalgstr* is one of the following:

endalgstr
is the label of a field that contains the algorithm string used to calculate the ordinal number.

A*l**endalgstr*
is the label of a 4-byte field that contains the storage address of the algorithm string used to calculate the ordinal number.

BEGORD=*bordnum*
specifies the begin ordinal number for use in a subsequent macro that specifies the FULLFILE parameter, where *bordnum* is one of the following:

bordnum
is the label of a field that contains the ordinal number.

A*l**bordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number.

Note: This parameter is provided only for migration purposes. Use the BEGALG parameter to specify the begin ordinal number.

ENDORD=*eordnum*
specifies the end ordinal number for use in a subsequent macro that specifies the FULLFILE parameter, where *eordnum* is one of the following:

eordnum
is the label of a field that contains the ordinal number.

A*l**eordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number.

Note: This parameter is provided only for migration purposes. Use the ENDALG parameter to specify the end ordinal number.

WRAPAROUND
reads LRECs from the start of the file to the end until it has read the whole file. Use this parameter value only when you intend to use fullfile processing.

For example, consider a file that contains five subfiles and the current subfile is number 3. If you specify the WRAPAROUND parameter, and then call a DBRED macro with the FULLFILE parameter, LRECs would be read from the subfiles in the order: 3, 4, 0, 1, 2.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming*

Macros for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

INTERLV=*intrlvnum*
specifies the number of the interleave that you want to use, where *interlvnum* is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN=*partitnum*
specifies the number of the partition that you want to use, where *partitnum* is one of the following:
- A register that contains the address of the partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm calculates the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this macro:
- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the NODUMP parameter is not recommended because it can prevent system errors from being issued that indicate a critical problem.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH=*pathnum*
specifies the path number for a detail subfile using index support, where *pathnum* is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator. If there is only one index path, do not specify this parameter.

See *TPFDF Database Administration* for more information about path numbers.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

None.

Normal Return

The DBADR macro does not change the current LREC even if you specify a different value for the ALG parameter with the DBADR macro from that which was used to locate the LREC.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.

Examples

- In the following example, the file is processed during fullfile processing between the ordinals calculated from the algorithm strings identified with the BEGALG and and ENDALG parameters.

```
DBADR REF=GR25DF,BEGALG==C'A',ENDALG==C'K'
DBRED REF=GR25DF,FULLFILE
```

- Assume file GR25DF has ordinals 0–5. The following example processes subfiles 5, 0, and 1 (in that order) in file GR25DF.

```
DBADR REF=GR25DF,ENDORD==F'1'
DBADR REF=GR25DF,BEGORD==F'5',WRAPAROUND
DBRED REF=GR25DF,FULLFILE
```

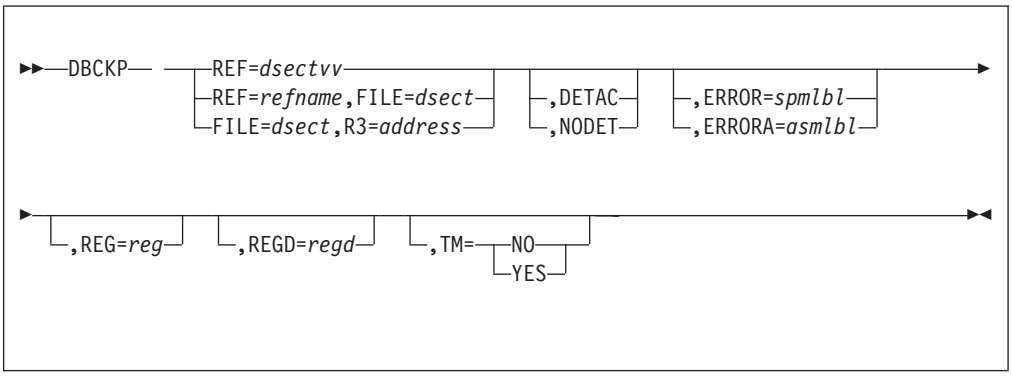
Related Macros

- “DBDEL–Delete One or More Logical Records” on page 214
- “DBDSP–Display Logical Records from a Subfile” on page 231
- “DBMOD–Perform or Indicate Logical Record Modifications” on page 249
- “DBMRG–Merge Logical Records from Two Subfiles” on page 254
- “DBRED–Read a Logical Record” on page 272
- “DBSRT–Sort a Subfile” on page 307
- “DBTLG–Write a File or Subfile to Tape” on page 318.

DBCKP–Checkpoint a Subfile

Use this macro to checkpoint a subfile; that is, all blocks in main storage that have been changed are copied to DASD.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

DETAC
places the subfile in detac mode after checkpointing the subfile. When the subfile is in detac mode, all modified blocks are saved in main storage. Any changes that you make to the LRECs in that subfile are not written to DASD until the subfile is checkpointed or closed. You can discard modified LRECs (prevent them from being written to DASD) by using the ABORT parameter of the DBCLS macro.

Note: The TPF system and the ALCS environment issues a 000010 system error if an application program does not give up control in the time allotted by the application time-out counter. When processing in detac

mode, a TPFDF application program can require more than the allotted time on a database with a large data structure. To prevent the 000010 system error, you can change the setting of the &TPFDBDV symbol in the DBLCL macro.

See *TPFDF Installation and Customization* for more information about the &TPFDBDV symbol and the DBLCL macro.

NODET
specifies that you do not want the subfile in detac mode after it has been checkpointed.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

TM
specifies one of the following:

NO
specifies that commit scopes are not used during checkpoint processing, regardless of what the database definition (DBDEF) macro has set as the default.

YES
specifies that commit scopes are used during checkpoint processing, regardless of what the DBDEF macro has set as the default. This option is valuable when many files are to be filed out during checkpoint processing (for example, detac mode, extensive B^+Tree indexing updates, and requests that result in packing).

Entry Requirements

None.

Normal Return

The subfile remains open and the current logical record (LREC) remains the same.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- If the DBCKP macro is issued for a W-type file, the file is written to DASD in short-term pool records.
- For a B^+Tree file, the DBCKP macro checkpoints the index blocks and the data blocks.
- Processing the DBCKP macro on a file that is opened in a commit scope is not visible until the file is committed. If the checkpointed file is rolled back, updates to the file are discarded. See “Commit Scopes” on page 8 for more information commit scopes.

Examples

The following example saves any updates made to the current subfile.

```
DBCKP REF=GR30SR,ERROR=BFAQER6
```

Related Macros

- “DBCLS–Close a Subfile” on page 199
- “DBOPN–Open a Subfile” on page 260.

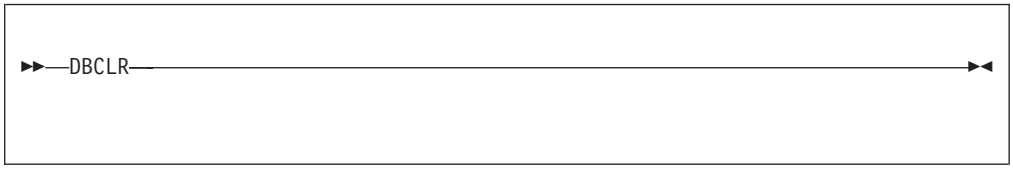
DBCLR–Allow ECB Exit with Open Files

Use this macro to allow a program to exit without closing any open subfiles and without generating a dump.

If you want to process an EXITC macro when a program still has subfiles open that you do not want to save, call the DBCLR macro first. The TPFDF product suppresses the dump that would normally be caused by such an action.

Attention: We do not recommend using this macro because it can leave subfiles in a partially updated condition. Instead, use the DBCLS macro with the ABORT parameter if you want to discard updates made while a subfile is in detac mode.

Format



Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.

Examples

The following example allows the program to exit with files still open.

```
DBCLR
```

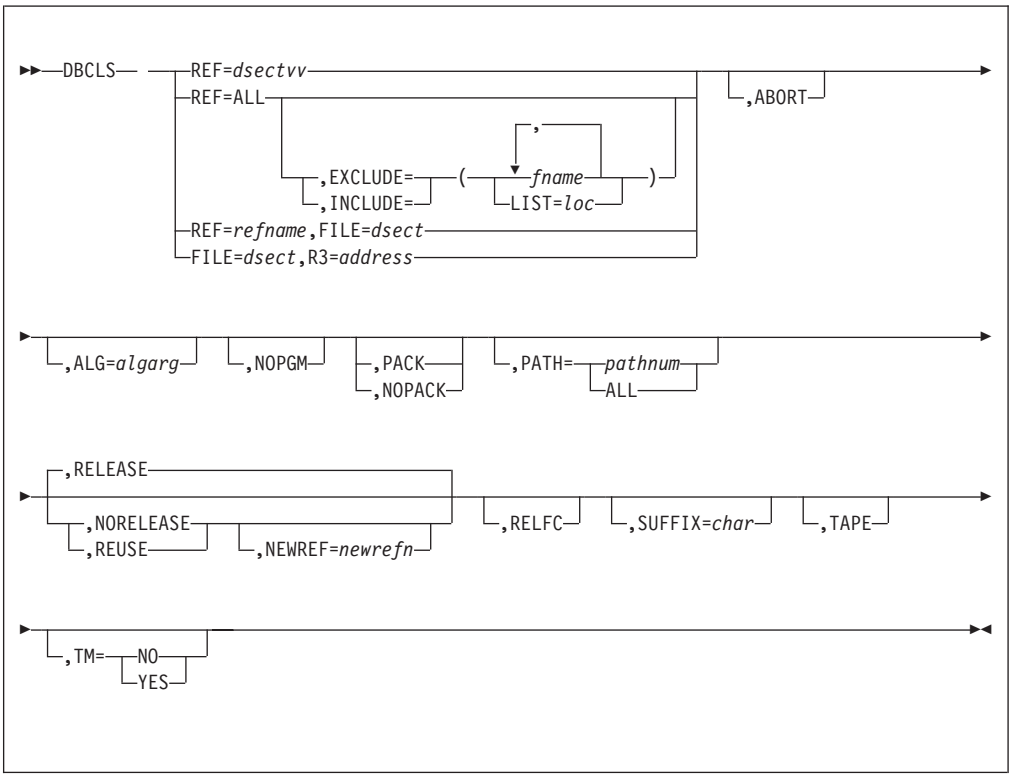
Related Macros

“DBCLS–Close a Subfile” on page 199.

DBCLS–Close a Subfile

Use this macro to close one or more subfiles. You can also choose whether you want to write modified blocks that are in detac mode to DASD.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

REF=ALL
closes all open subfiles. If you specify REF=ALL, only those subfiles that you opened using the DBOPN macro without the R3 parameter specified are closed.

Note: When you use this parameter, all database interface blocks (DBIFBs), which contain SW00SR slots, are released. However, DBIFBs are ***not*** released if there are no files open or any of the following parameters have been specified:

- EXCLUDE

- INCLUDE
- NORELEASE
- REUSE.

EXCLUDE
closes all files ***except*** those specified with this parameter.

INCLUDE
closes ***only*** the files specified with this parameter. If any of the listed files are not open, they are ignored.

fname
is the reference name of the file or files that you want to exclude or include in the DBCLS macro processing.

LIST=*loc*
specifies a list of files to be included in or excluded from the DBCLS macro processing, where *loc* is one of the following:

- A register that contains the address of the list of files
- A label in one of the following formats:

  loc
  is the label of a field that contains the list of files.

  A*loc*
  is the label of a 4-byte field that contains the storage location of the list of files.

For example:

```
DBCLS REF=ALL,EXCLUDE=(LIST=GR25LST)
```

The specified location must start with a halfword stating the number of files listed, followed by a list of 8-byte reference names.

For example, GR25LST can contain the following data:

```
DC   H'3'
DC   C'GR25DF  '
DC   C'GR25DA  '
DC   C'GR25DB  '
```

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ABORT
causes all database updates since the file was opened, or since the last DBCKP macro, to be discarded. The updates are not written to DASD.

Note: Use this option only if the subfile is in detac mode.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

NOPGM
specifies not to change the program stamp in a block when filing it.

PACK
packs the subfile.

NOPACK
specifies that you do not want to pack the subfile, even if a block has fallen below the packing threshold defined by the PIN parameter on the DBDEF macro. See *TPFDF Database Administration* for more information about the packing threshold.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

RELEASE
releases the SW00SR slot for the subfile after the macro has completed processing.

NORELEASE
prevents the SW00SR slot from being released when the file is closed. Any key parameters you have defined are also retained. Specify this parameter if you intend to process the same subfile at a later time.

REUSE
retains the SW00SR slot of the file. Any key parameters you have defined are also retained. Specify this parameter if you intend to retrieve another subfile in the same file at a later time.

NEWREF=*newrefn*
changes the reference name of the file specified with the REF parameter, where *newrefn* is one of the following:

- An explicit term that represents the new reference name
- A label in one of the following formats:

  newrefn
  is the label of an 8-byte field that contains the new reference name.

  A*I**newrefn*
  is the label of a 4-byte field that contains the storage address of the 8-byte field containing the new reference name.

RELFC
releases the subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

W-type files are automatically released unless they have been sorted, merged, or checkpointed. In these cases, you must specify the RELFC parameter to release W-type files.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

TAPE
closes the tape or sequential data set at the same time as any subfiles are closed. You must specify this parameter if you specified the TAPE parameter on a previous DBOPN macro.

Note: Do not use the TAPE parameter with the TM parameter specified with a value of YES because the integrity of the commit scope could be compromised, and files that are on tape cannot be rolled back.

TM
specifies one of the following:

NO
specifies that commit scopes are not used during close processing, regardless of what the database definition (DBDEF) macro has set as the default.

YES
specifies that commit scopes are used during close processing, regardless of what the DBDEF macro has set as the default. This option is valuable when many files are to be filed out during close processing (for example, detac mode, extensive B^+Tree indexing updates, and requests that result in packing).

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- Do not check the error indicators in the SW00RTN field of the SW00SR slot. When you close a subfile, the DBCLS macro releases the SW00SR slot for the file (unless you specify the NORELEASE or REUSE parameter), so the error indicators are not available for you to check. When you specify the NORELEASE or REUSE parameter, the SW00SR slot is not released but the error indicators are not valid.
- Deleting LRECs from a subfile leaves empty space in the blocks. If you do not specify the PACK or NOPACK parameter and the file is not a B^+Tree file, the subfile is packed if the amount of space occupied by LRECs in any block falls below the threshold defined in the database definition (DBDEF).

  B^+Tree files are not packed during close processing unless the PACK parameter is specified or there are no nodes in the B^+Tree index.
- It is not necessary to close T-type files. In addition, it is not necessary to close W-type files that are open in detac mode. See *TPFDF Database Administration* for more information about T-type and W-type files.
- If you specify the RELFC parameter on the DBCLS macro, the TPFDF product issues an internal DBDIX macro.
- If you do not specify the INCLUDE and REF=ALL parameters and an attempt is made to close a subfile that is not open, the TPFDF product issues a DB0115 system error.
- If you specify both the REF=ALL and TM=YES parameters when there are multiple files open, the files in a commit scope are processed individually.

Examples

- The following example deletes all the LRECs in the subfile and any corresponding index LRECs, and releases the subfile. Any overflow blocks are released and the prime block is initialized to empty (for a fixed file) or released (for a pool file).

  ```
  DBCLS REF=GR23DF,RELFC,ALG=EBW044
  ```

Note: This example has the same effect as the following sequence of macro calls:

```
DBRED REF=GR23DF,ALG=EBW044
DBDEL REF=GR23DF,ALL,NOKEY
DBDIX REF=GR23DF,ALG=EBW044
DBCLS REF=GR23DF
```

- The following example shows how you can use a different ALG parameter to access LRECs in another subfile belonging to the same file without calling another DBOPN.

```
DBOPN REF=GR22DF,ALG==C'A'
DBCLS REF=GR22DF,REUSE
DBRED REF=GR22DF,ALG==C'B'
```

- The following example closes only two files, GR25DA and GR25DB. If GR25DA or GR25DB (or both) are not open, the files that are not open are ignored.

```
DBCLS REF=ALL,INCLUDE=(GR25DA,GR25DB)
```

- The following example closes all files ***except*** the two files, GR25DA and GR25DB.

```
DBCLS REF=ALL,EXCLUDE=(GR25DA,GR25DB)
```

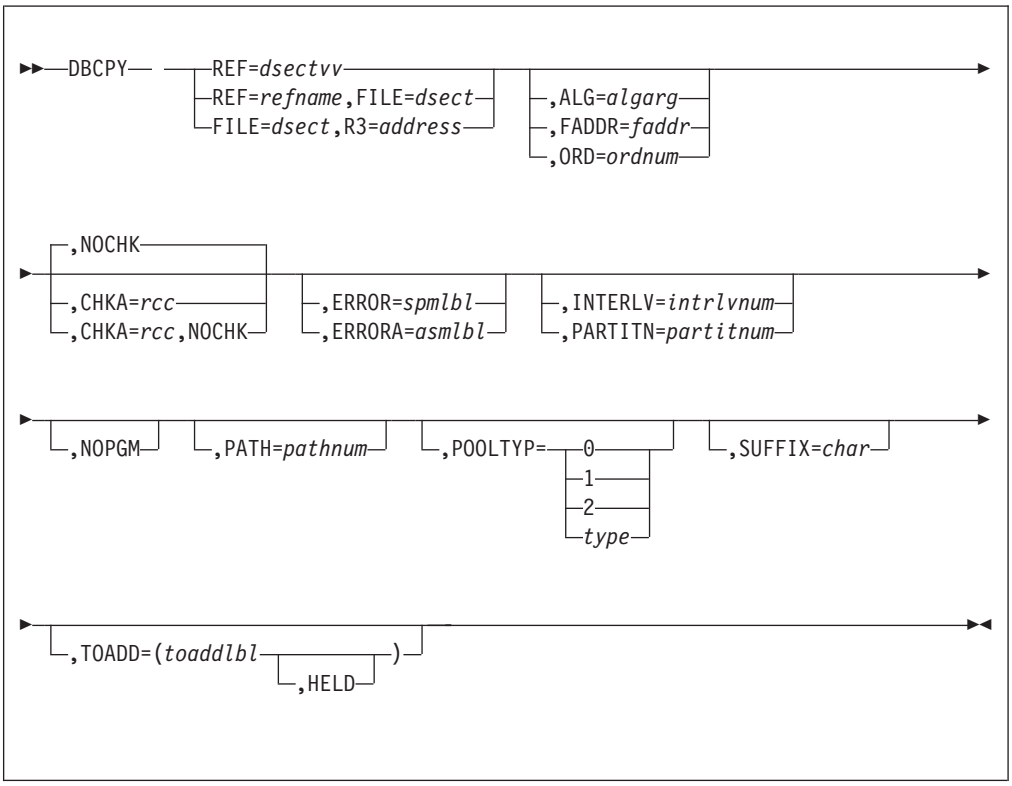
Related Macros

- “DBOPN–Open a Subfile” on page 260
- “DBCKP–Checkpoint a Subfile” on page 195.

DBCPY–Copy a Subfile

Use this macro to copy a subfile. After executing the command, the TPFDF product closes the subfile and performs all subsequent actions on the copy.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

INTERLV=*intrlvnum*
specifies the number of the interleave that you want to use, where *interlvnum* is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN=*partitnum*
specifies the number of the partition that you want to use, where *partitnum* is one of the following:
- A register that contains the address of the partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm calculates the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH=*pathnum*
specifies the path number for a detail subfile using index support, where *pathnum* is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator. If there is only one index path, do not specify this parameter.

See *TPFDF Database Administration* for more information about path numbers.

POOLTYP
overrides the pool type defined by the database administrator, where:

0 uses the pool type defined by the PF0 parameter of the DBDEF macro.

1 uses the pool type defined by the PF1 parameter of the DBDEF macro.

2 uses the pool type defined by the PF2 parameter of the DBDEF macro.

type
is the label of a 1-byte field that contains a 0, 1, or 2 to specify to pool type.

Use the POOLTYP parameter as directed by the database administrator.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

TOADD=*toaddlbl*
copies the subfile to a specified prime block, where *toaddlbl* is a 4-byte field containing the address of the prime block.

If you want to create a new pool subfile, do one of the following:

- Do not specify this parameter
- Specify an address of hexadecimal zeros for this parameter.

If you specify the TOADD parameter, the new subfile overwrites any data that is already there.

If used in a commit scope, the prime block specified by the TOADD parameter must be opened in the same commit scope as the subfile that is being copied. See “Commit Scopes” on page 8 for more information about commit scopes.

HELD
specifies that the entry control block (ECB) is already holding the file address specified by the TOADD parameter.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.

- The following rules determine the value of the record code check (RCC) value used when the TPFDF product creates a new subfile:
  - If you do not specify the NOCHK parameter, the TPFDF product creates new subfiles with a random RCC value.
  - If you specify the NOCHK parameter without the CHKA parameter, the TPFDF product creates new subfiles without an RCC value.
  - If you specify both the NOCHK and CHKA parameters, the TPFDF product creates new subfiles with the RCC value specified with the CHKA parameter.
- When a B+Tree file is copied, the DBCPY macro builds the B+Tree index for the new file.

Examples

The following example copies a subfile to the file address specified at label EBX040. This file address is already being held by the ECB.

```
DBCPY REF=GR30SR,TOADD=(EBX040,HELD),POOLTYP=1
```

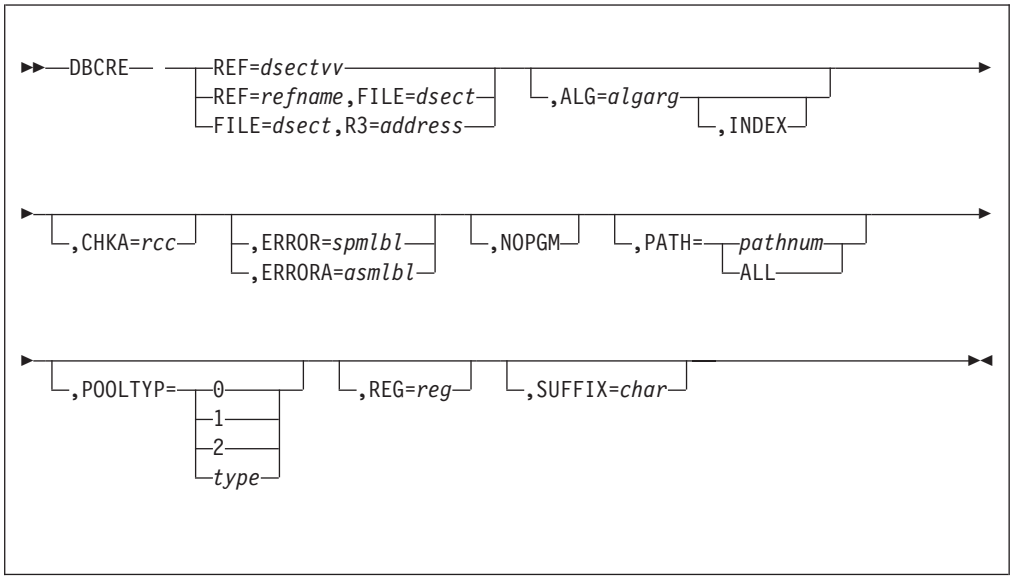
Related Macros

"DBRST–Restore a Subfile" on page 293.

DBCRE–Create a Subfile

Use this macro to create a new subfile, an empty pool subfile, or an empty indexed pool subfile with its corresponding index file. You can subsequently add logical records (LRECs) to the empty detail subfile as required.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

INDEX
creates an indexed subfile and inserts an index LREC referencing this subfile into the related index file (or files) defined by the database administrator. Specify the index key as the ALG parameter.

If you specify this parameter, the algorithm defined for the new subfile must be #TPFDBFF.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

POOLTYP
overrides the pool type defined by the database administrator, where:

0 uses the pool type defined by the PF0 parameter of the DBDEF macro.

1 uses the pool type defined by the PF1 parameter of the DBDEF macro.

2 uses the pool type defined by the PF2 parameter of the DBDEF macro.

type
is the label of a 1-byte field that contains a 0, 1, or 2 to specify to pool type.

Use the POOLTYP parameter as directed by the database administrator.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

None.

Normal Return

The address of the LREC created is placed in the SW00REC field of the SW00SR. If you specify the REG parameter, the address of the LREC created is placed in the specified register.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.

- A DBADD macro creates a subfile, if one does not exist, before it adds the LREC.

Examples

- The following example creates an indexed subfile and adds a new LREC to that subfile.

```
DBCRE REF=GR45DF,ALG=EBW001,INDEX
DBADD REF=GR45DF,NEWLREC=EBW000
```

- The following example creates a new subfile, creates the index reference to that subfile, and adds an LREC to the subfile.

```
DBCRE REF=GR23DF
DBIDX REF=GR23DF,ALG=EBW0044
DBADD REF=GR23DF
```

Related Macros

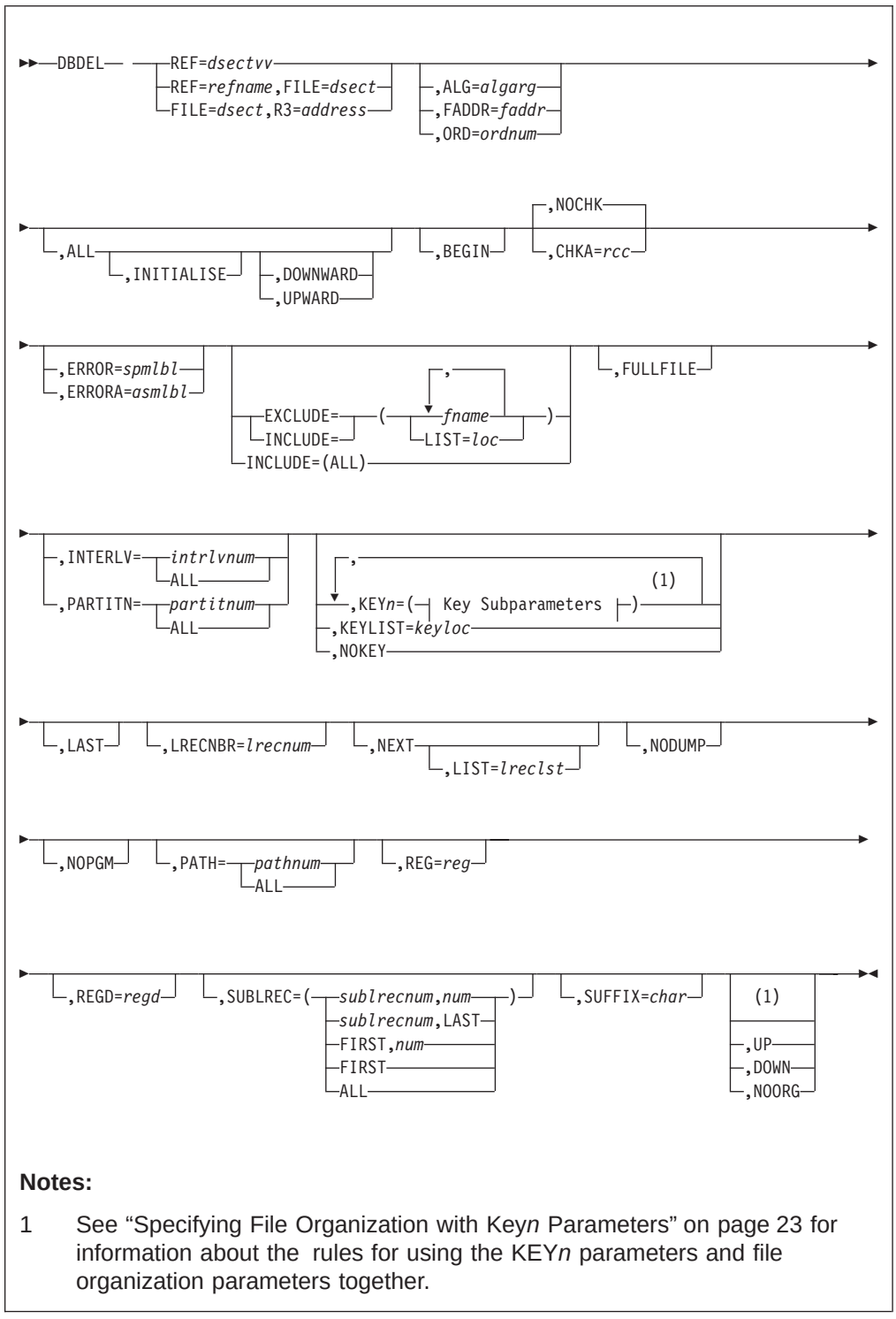
- “DBADD–Add a Logical Record to a Subfile” on page 176
- “DBDIX–Delete Index References to a Subfile” on page 228
- “DBIDX–Create an Index Reference” on page 241.

DBDEL–Delete One or More Logical Records

Use this macro to delete:

- One or more logical records (LRECs) from an open subfile
- The LRECs in one or more subfiles of a file
- A complete extended LREC
- All of the LRECs in a file
- Some or all of the subfiles referenced from the LRECs that you delete
- One or more subLRECs in an extended LREC.

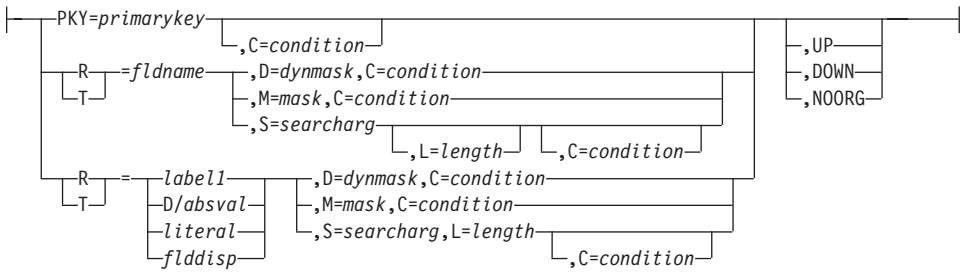
Format



Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

Key Subparameters:



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AL*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument

- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

ALL
deletes every LREC in the open subfile specified by the REF parameter. If you opened the subfile using KEY*n* parameters, the DBDEL macro deletes only the LRECs that match these keys.

If you delete all the LRECs from a fixed file, the DBDEL macro writes the empty prime block to DASD after deleting the LRECs (the block header and optional trailer are not deleted). Any blocks previously chained to the prime are released.

If you delete all the LRECs from a pool subfile, the DBDEL macro releases both prime and overflow blocks. However, if you delete all the LRECs from a pool subfile, and add an LREC before you close the subfile, the prime block is not released.

For B^+Tree files, if no keys are specified, the DBDEL macro with the ALL parameter releases the index blocks as well as the data blocks.

If the subfile was opened in detac mode, you cannot recover the subfile using the DBCLS macro with the ABORT parameter specified.

If you specify the FULLFILE parameter in addition to the ALL parameter, the DBDEL macro deletes every LREC in every subfile of the file. You can delete

LRECs in certain subfiles only by specifying the BEGORD and ENDORD parameters with the DBOPN or DBADR macros.

If you specify KEY*n* parameters in addition to ALL, DBDEL deletes all the LRECs in the open subfile that match these keys. If you also specify the FULLFILE parameter, the DBDEL macro deletes all the LRECs that match the keys in an entire file. In this case, the blocks are not released, whether they are fixed or pool, because some LRECs can remain in a file after deletion.

You can use the ALG parameter with the ALL parameter to delete LRECs from an indexed detail file. The DBDEL macro deletes the index entry for the subfile and releases the indexed subfile if it is in pool.

INITIALISE
empties an entire subfile, apart from the standard TPFDF header in the prime block. The TPFDF product initializes the subfile (it deletes all LRECs, but leaves the empty prime block) and releases any blocks previously chained to the prime block.

Note: If you specify the INITIALISE parameter with the ALL and KEY*n* parameters, the INITIALISE parameter is ignored.

DOWNWARD
deletes all LRECs from (and including) the current LREC to the last LREC in the subfile. If you opened the subfile using KEY*n* parameters, the DBDEL macro deletes only the LRECs that match these keys.

UPWARD
deletes all LRECs from (but not including) the current LREC to the first LREC in the subfile. If you opened the subfile using KEY*n* parameters, the DBDEL macro deletes only the LRECs that match these keys.

BEGIN
searches from the beginning of the subfile for LRECs to delete.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

EXCLUDE
prevents the deletion of specific subfiles that are referenced from the LREC that is being deleted.

INCLUDE
deletes specific subfiles that are referenced by deleted LRECs.

ALL
deletes all subfiles referenced by the LRECs that you are deleting. You can also use this parameter with the ALL or FULLFILE parameter.

Notes:

1. With INCLUDE=(ALL), if the deleted subfiles contain LRECs that reference other subfiles, the DBDEL macro also deletes these referenced subfiles. This process continues to as many levels as necessary.
2. If the subfile containing the LREC being deleted was opened in detac mode, the subfiles referenced by the LREC cannot be recovered using DBCLS ABORT.

Attention: Do not use the INCLUDE parameter with subfiles that you have opened in detac mode. The TPFDF product deletes any referenced subfiles when you issue the DBDEL macro. If you subsequently close the subfile using an ABORT parameter, the subfile will contain LRECs with references to nonexistent subfiles.

fname
is the name of the subfile or subfiles that you want to exclude or include in the DBDEL macro processing. You can specify as many as 10 DSECT names.

LIST=*loc*
specifies a list of subfiles that you want to include or exclude in the DBDEL macro processing, where *loc* is the location of a 22-byte field containing a list of as many as 10 file IDs. Specify one of the following:

loc
is the label of the field that contains the list.

A*l**oc*
is the label of a 4-byte field containing the storage address of the field that contains the list.

The list starts with a 2-byte field that contains a halfword count of the number of file IDs referenced, each subsequent 2-byte field contains a file ID.

Note: The location contains the file IDs, not the file names.

FULLFILE
deletes an LREC specified with the ALG and KEY*n*parameters from any of the subfiles in the file referenced by the REF parameter. If you do not specify the FULLFILE parameter, you can only delete an LREC in the current subfile.

Note: If you specified the BEGORD and ENDORD parameters with the DBOPN macro, the DBDEL macro only deletes the subfiles between the specified ordinals.

INTERLV
specifies the interleave that you want to use. Specify one of the following:

interlvnum
is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

ALL
specifies all interleaves. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different interleave.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN
specifies the partition that you want to use. Specify one of the following:

partitnum
is one of the following:
- A register that contains the address of partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

ALL
specifies all partitions. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different partition.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm computes the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

KEY*n*
specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See “Specifying File Organization with Key*n* Parameters” on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

PKY=*primarykey*
specifies a value that will be compared against the primary key of an LREC, where *primarykey* is a 1-byte immediate value; for example:

```
... KEY1=(PKY=#RR00K80)
```

This has the same effect as:

```
... KEY1=(R=RR00KEY,S=#RR00K80)
```

R specifies a field in the LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

T specifies a field in the subLREC of an extended LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,S=EBW000)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=EBX010,S=EBW000,L==H'4')
```

D*I**absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/2,S=EBW000,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/GR00NAM-GR00REC,S=EBW000,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R==H'2',S=EBW000,L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+2,S=EBW000,L==H'4')
```

or

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+L'GR00FLD,S=EBW000,L==H'4')
```

C=*condition*
specifies the condition to be used when comparing fields in the logical record (specified with the R subparameter) with the search argument (specified with the S or PKY subparameter) or with the bit mask (specified with the M or D subparameter).

If you specify the S or PKY subparameter, use one of the following values:

Value	Condition
EQ	Equal (this is the default)
E	Equal
NE	Not equal
GE	Greater than or equal
LE	Less than or equal
GT	Greater than
LT	Less than
H	High
L	Low
NH	Not high
NL	Not low.

If you specify the M or D subparameter, use one of the following values:

Value	Condition
Z	Zeros
O	Ones
M	Mixed
NZ	Not zeros
NO	Not ones
NM	Not mixed.

D=*dynmask*
specifies the label of a 1-byte field containing a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,D=EBW000,C=Z)
```

M=*mask*
specifies a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,M=X'80',C=Z)
```

S=*searcharg*
specifies the search argument to be compared with the LREC field specified with the R or T subparameter, where *searcharg* is one of the following:

- A register that contains the address of the search argument
- A literal that represents the search argument
- A label in one of the following formats:

  searcharg
  is the label of the search argument.

  A/*searcharg*
  is the label of a 4-byte field that contains the storage address of the search argument.

  P/*searcharg*
  is the label of a field that contains the search argument in packed decimal format.

  If you specify P/*searcharg* or a literal in the form of =P"...", the LREC field and search argument are compared as decimal numbers in packed format. Otherwise, the LREC field and search argument are compared as character data.

L=*length*
specifies the length of the search argument, where *length* is one of the following:

- The address of a 2-byte field containing the length of the search argument
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

NOORG
specifies that the key field is in no particular order in the subfile.

KEYLIST=*keyloc*
specifies a key list that you want to use with this macro, where *keyloc* is one of the following:

- A register that contains the address of the key list
- A label in one of the following formats:

  keyloc
  is a label indicating the address of the key list.

AI*keyloc*
is the label of a 4-byte field that contains the storage address of the key list.

See “Setting Up and Using a Key List” on page 26 for information about how to set up a key list.

NOKEY
deactivates any currently active keys.

Notes:

1. If the file is not open when you specify the NOKEY parameter, the DBDEL macro opens the file and deletes the *first* LREC.
2. If the file is open and you specify the REF and NOKEY parameters, the DBDEL macro deletes the *next* LREC.
3. If the file is open and you specify the FILE and NOKEY parameters, the DBDEL macro deletes the *current* LREC.

LAST
deletes the last LREC in a subfile.

If you specify KEY*n* parameters as well, the DBDEL macro deletes the last LREC that matches these keys.

LRECNBR=*lrecnum*
specifies the sequence number of an LREC that you want to access, where *lrecnum* is one of the following:

- A register that contains the address of the LREC number
- An immediate value that represents the LREC number
- A label in one of the following formats:

  lrecnum
  is the label of a 4-byte field that contains the LREC number.

  AI*lrecnum*
  is the label of a 4-byte field that contains the storage address containing the LREC number.

Notes:

1. If you use the #TPFDB0D algorithm, you must specify this parameter.
2. LRECs are numbered in increasing order from the start of the subfile (the first LREC in the prime block has sequence number 1).
3. If you specify the LRECNBR parameter with KEY*n* parameters, only those LRECs that match the key conditions are included in the sequence numbering; LRECs that do not match are ignored.

NEXT
steps through and deletes LRECs from a subfile. If you do not specify any subparameters with the NEXT parameter, the DBDEL macro deletes the next LREC in the subfile after the current LREC.

LIST=*lreclst*
deletes a list of LRECs following the current LREC, where *register* is a register that contains the address of the list of the LRECs. The list contains one or more LREC sequence numbers separated by a slash (/). You can also specify a range of sequence numbers by separating the beginning and end of the range by a hyphen (-). You can use LAST to mean the last LREC of the subfile and ALL to mean all the remaining LRECs. You can also end the list with a nonnumeric character.

Notes:

1. The ranges must be in ascending order; if one is found out of order, that range and all subsequent ranges are ignored.

   For example, if there are 41 LRECs in a subfile, the following lists all have the same effect:

   ```
   20/31/32/33/37/38/39/40/41
   20/31/32/33/37-41
   20/31-33/37-LAST
   20/31-33/37/ALL
   ```

2. You ***cannot*** specify the number zero in the list of LREC numbers, even if you specify the ADJUST parameter with a value that would adjust the number zero to a valid LREC number.

NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this macro:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the NODUMP parameter is not recommended because it can prevent system errors from being issued that indicate a critical problem.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

SUBLREC
deletes some or all of the subLRECs in an extended LREC. Use one of the following to specify the range of subLRECs you want to delete:

sublrecnum
is the number of the subLREC from which you want to begin deleting.

Note: The first subLREC in an extended LREC is 0.

num
is the number of subLRECs you want to delete.

FIRST
deletes the first subLREC (the latest one added to the extended LREC).

LAST
deletes the subLRECs to the end of the extended LREC (until the first subLREC added to the extended LREC).

ALL
deletes every subLREC in the LREC you have specified.

If you specify the ALL parameter (not as a subparameter of the SUBLREC parameter), the TPFDF product deletes the requested subLRECs in all the LRECs in the open subfile. For example, the following instruction deletes the first subLREC in every extended LREC in a subfile:

```
DBDEL REF=GR24DF,SUBLREC=(FIRST),ALL
```

Note: When you specify the SUBLREC parameter, do not specify any of the access parameters (ALG, ORD, FADDR).

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

NOORG
specifies that the LRECs are organized in the subfile in no particular order. (NOORG is the default if subfile organization has not been defined in the DBDEF).

Entry Requirements

Before specifying the SUBLREC parameter, you must specify a DBRED macro to locate the extended LREC from which you want to delete the subLREC.

Normal Return

- Field SW00REC contains one of the following:
  - A pointer to the next LREC in the subfile (after the last deleted LREC)
  - A pointer to the current extended LREC from which the subLREC or subLRECs were deleted.
- After you use the ALL parameter with the DOWNWARD or UPWARD parameter, the SW00REC is set to zero and the SW00RTN is set to X'40'.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.
- If you use the LIST parameter, check bit 0 in SW00RT2 in addition to SW00RTN.

- If you use the FULLFILE parameter, check the error count in the SW00RT1 field because the SW00RTN field has only an end-of-file indicator set.

  If you use the FULLFILE parameter, and the end-of-file indicator is set, you cannot issue additional TPFDF macros until the file is closed. However, you can specify the REUSE parameter on the DBCLS macro. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- If you specify the SUBLREC parameter, there is no error return when the subLRECs that you specify to be deleted do not exist. In all cases, SW00RTN contains X'00' as if the command processed successfully.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- You cannot specify six KEY*n* parameters (KEY1–KEY6) when both of the following conditions are true:
  - The S subparameter is used on all six keys
  - The length of all keys is greater than 1 (either with an explicit length using the L subparameter or an implied length from the field used in the R or T subparameter).

  If you need this type of key definition, you must use a key list.
- Any active keys are ignored when you use the #TPFDB0D algorithm.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent macros when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with macros that access or update records without using fullfile processing.
- If you do not specify any optional parameters, the DBDEL macro deletes the current LREC (usually the last LREC read) from the subfile that is currently open.
- If block index support has been defined with a file, DBDEL modifies the block index of the blocks from which it deletes LRECs.
- If you specify a KEY*n* parameter or the LRECNBR parameter, a NEXT parameter is implied, so do not specify NEXT explicitly.
- When you use DBDEL to delete subLRECs, the control fields in the extended LREC are automatically adjusted.
- If you specify the ALL parameter for a P-type file, the whole file is deleted.

  Note: Other DBDEL parameters are not supported for P-type files.
- If you specify the ALL and NOKEY parameters for a B^+Tree file, the DBDEL macro will pack the file.
- Before you use the ALL parameter with the DOWNWARD or UPWARD parameter, first establish a *current LREC* (for example, using the DBRED macro).

Attention: Using the ALL parameter with the DOWNWARD or UPWARD parameter can cause the values that were created previously for the DBRET macro to be corrupted.

Examples

- The following example deletes all the LRECs in the open subfile that match the specified keys.

```
DBDEL REF=GR22DF,ALL,KEY1=(R=GR22ALN,S=A/EBW008)
```

- The following example deletes all the LRECs in file GR22DF.

```
DBDEL REF=GR22DF,INITIALISE,ALL
```

- The following example deletes the next LREC in open subfile GR22DF, together with all subfiles referenced from this deleted LREC.

```
DBDEL REF=GR22DF,NEXT,INCLUDE=(ALL)
```

- The following example deletes all the LRECs in open subfile GR22DF, together with all subfiles referenced from any deleted LRECs.

```
DBDEL REF=GR22DF,ALL,INCLUDE=(ALL)
```

- The following example deletes all LRECs in open subfile GR22DF, together with any subfiles with file ID X'FE30' or X'FE31' referenced from any delete LRECs.

```
MVC   EBX000(2),=H'2'                 Count of file IDs in list
MVC   EBX002(2),=X'FE30'              Add file ID to list
MVC   EBX004(2),=X'FE31'              Add file ID to list
DBDEL REF=GR22DF,ALL,INCLUDE=(LIST=EBX000)
```

- The following example deletes all LRECs in all subfiles in file GR22DF, together with all subfiles referenced from any of these LRECs.

```
DBDEL REF=GR22DF,FULLFILE,ALL,INCLUDE=(ALL)
```

- The following example deletes subfiles GR21SR and GR22SR, but not GR23SR (assuming all three subfiles are referenced from LRECs in subfile GR20SR).

```
DBDEL REF=GR20SR,ALL,INCLUDE=(GR21SR,GR22SR),EXCLUDE=(GR23SR)
```

- The following example deletes the LREC specified by the 4-byte LREC number contained in register 4.

```
DBDEL REF=GR35DF,LRECNBR=R4
```

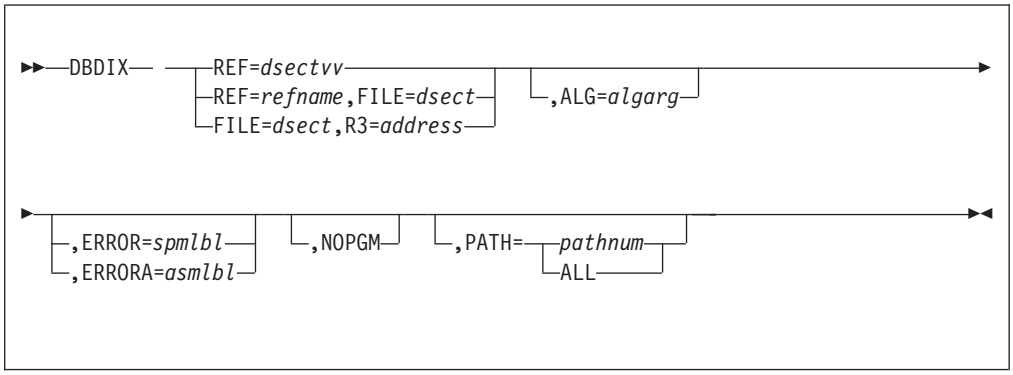
Related Macros

- “DBCLS–Close a Subfile” on page 199
- “DBMOD–Perform or Indicate Logical Record Modifications” on page 249
- “DBRED–Read a Logical Record” on page 272.

DBDIX–Delete Index References to a Subfile

Use this macro to delete index references to the current subfile or a specific subfile. This macro removes the reference to a detail subfile without deleting the LREC in the detail file.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

Entry Requirements

Ensure that the relationship of the index file (or index files, if there are multilevel indexes) to the detail file have been defined on the DBDEF macro by your database administrator.

Normal Return

Fields SW00RTN and SW00RT2 are set to 0.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- Using the DBDIX macro can result in having a detail file with no corresponding index file pointing to it.
- If the index subfile LREC is empty after processing, the TPFDF product releases the block and deletes any reference to it in a higher level index file.

  The top-level index file is fixed, so the TPFDF product does not release prime blocks in this file even if no index LRECs remain in it.
- If you specify the RELFC parameter on the DBCLS macro, the TPFDF product does an internal DBDIX macro.

Examples

The following example deletes all the LRECs in the subfile and any corresponding index LRECs, and releases the subfile. Any chained blocks are released and the prime block is initialized to empty (for a fixed file).

```
DBRED REF=GR23DF,ALG=EBW044
DBDEL REF=GR23DF,ALL,NOKEY
DBDIX REF=GR23DF,ALG=EBW044
DBCLS REF=GR23DF
```

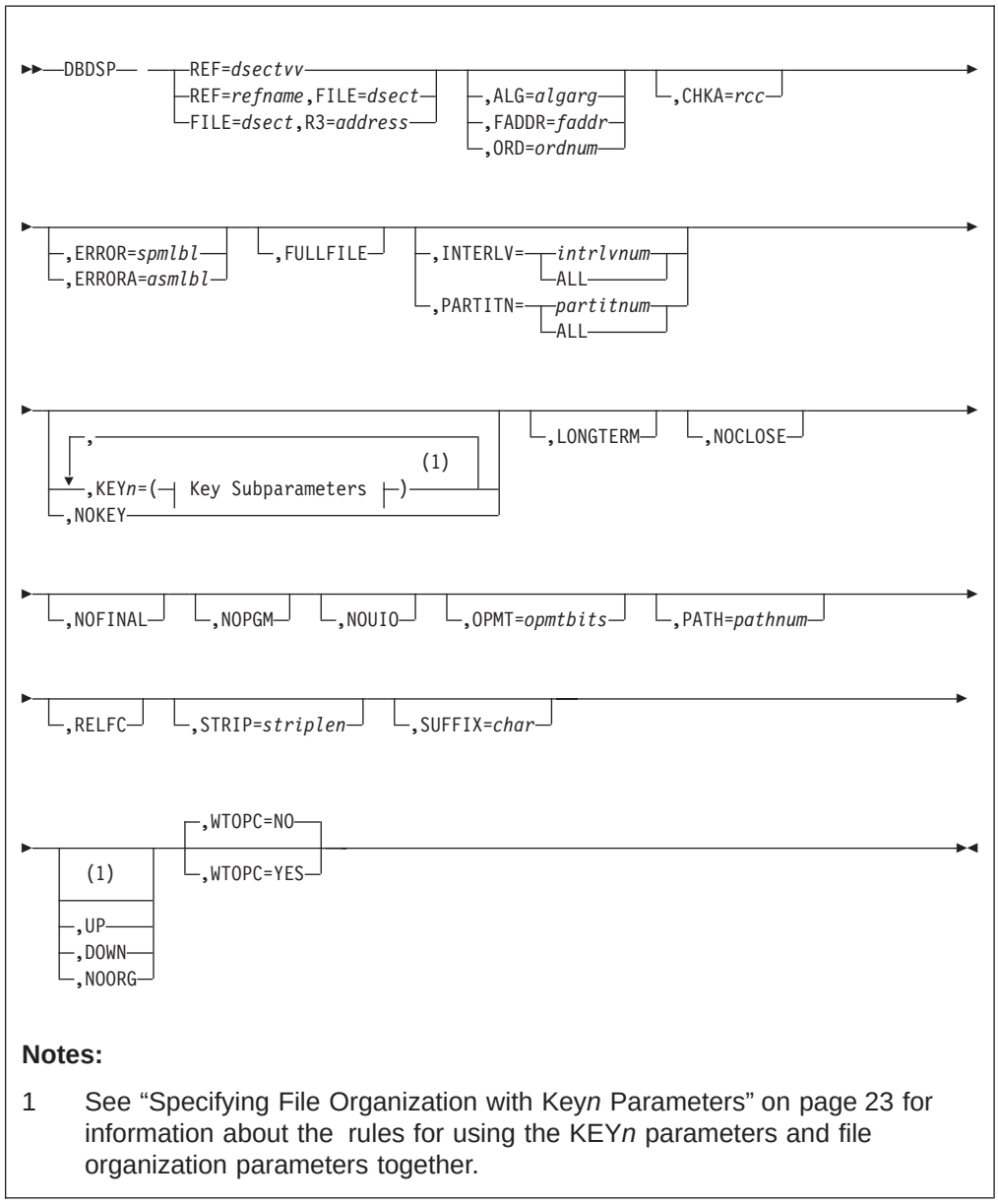
Related Macros

- “DBCLS–Close a Subfile” on page 199
- “DBCRE–Create a Subfile” on page 210
- “DBIDX–Create an Index Reference” on page 241.

DBDSP–Display Logical Records from a Subfile

Use this macro to display the logical records (LRECs) from a subfile.

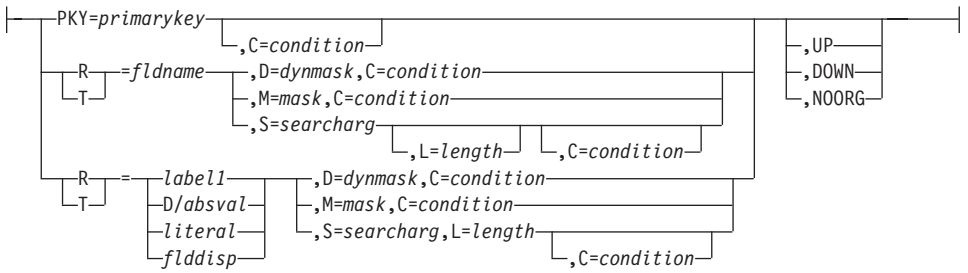
Format



Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

Key Subparameters:



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

A*I**refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument

- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FULLFILE
allows you to display LRECs from the whole file instead of from just one subfile. Do not use this parameter with W-type files or the NOCLOSE parameter.

INTERLV
specifies the interleave that you want to use. Specify one of the following:

interlvnum
is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

ALL
specifies all interleaves. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different interleave.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN
specifies the partition that you want to use. Specify one of the following:

partitnum
is one of the following:
- A register that contains the address of partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

ALL
specifies all partitions. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different partition.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm computes the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

KEY*n*
specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See “Specifying File Organization with Key*n* Parameters” on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

PKY=*primarykey*
specifies a value that will be compared against the primary key of an LREC, where *primarykey* is a 1-byte immediate value; for example:

```
... KEY1=(PKY=#RR00K80)
```

This has the same effect as:

```
... KEY1=(R=RR00KEY,S=#RR00K80)
```

R specifies a field in the LREC to be compared with the search argument

specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

T specifies a field in the subLREC of an extended LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,S=EBW000)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=EBX010,S=EBW000,L==H'4')
```

D*absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/2,S=EBW000,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/GR00NAM-GR00REC,S=EBW000,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R==H'2',S=EBW000,L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+2,S=EBW000,L==H'4')
```

or

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+L'GR00FLD,S=EBW000,L==H'4')
```

C=*condition*
specifies the condition to be used when comparing fields in the logical record (specified with the R subparameter) with the search argument (specified with the S or PKY subparameter) or with the bit mask (specified with the M or D subparameter).

If you specify the S or PKY subparameter, use one of the following values:

Value	Condition
EQ	Equal (this is the default)
E	Equal
NE	Not equal
GE	Greater than or equal
LE	Less than or equal
GT	Greater than
LT	Less than
H	High
L	Low
NH	Not high
NL	Not low.

If you specify the M or D subparameter, use one of the following values:

Value	Condition
Z	Zeros
O	Ones

M Mixed
NZ Not zeros
NO Not ones
NM Not mixed.

D=*dynmask*
specifies the label of a 1-byte field containing a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,D=EBW000,C=Z)
```

M=*mask*
specifies a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,M=X'80',C=Z)
```

S=*searcharg*
specifies the search argument to be compared with the LREC field specified with the R or T subparameter, where *searcharg* is one of the following:

- A register that contains the address of the search argument
- A literal that represents the search argument
- A label in one of the following formats:

  searcharg
  is the label of the search argument.

  A/*searcharg*
  is the label of a 4-byte field that contains the storage address of the search argument.

  P/*searcharg*
  is the label of a field that contains the search argument in packed decimal format.

  If you specify P/*searcharg* or a literal in the form of =P"...", the LREC field and search argument are compared as decimal numbers in packed format. Otherwise, the LREC field and search argument are compared as character data.

L=*length*
specifies the length of the search argument, where *length* is one of the following:

- The address of a 2-byte field containing the length of the search argument
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

NOORG
specifies that the key field is in no particular order in the subfile.

NOKEY
deactivates any currently active keys.

LONGTERM
instructs the application program to use the MOSG internal program to build the output message (OMSG) display using long-term pool records. If you do not specify this value, short-term pool records are used for the display by the FMSG program.

NOCLOSE
specifies that you do not want to close the subfile displayed with the DBDSP macro. This allows the application program to return to the open subfile once the macro has completed processing. If you specify this parameter, you cannot specify the FULLFILE parameter. In addition, if you specify the NOCLOSE parameter, ensure that you specify control be returned to the application program after the DBDSP macro processes.

NOFINAL
indicates that this is only part of a message. The complete output message is displayed only when you code the DBDSP macro without the NOFINAL parameter specified.

NOPGM
specifies not to change the program stamp in a block when filing it.

NOUIO
prevents the activation of the output edit CRT driver (UIO) and returns to the application program.

OPMT=*opmtbits*
specifies how you want the FMSG program to format the output message, where *opmtbits* is the value of the bit settings as defined in the UI2PF DSECT (labels UI2INC–UI2CNN). Specify one of the following:

opmtbits
is the label of a 5-byte field that contains the bit settings.

AI*opmtbits*
is the label of a 4-byte field that contains the storage address of the bit settings.

If you do not specify this parameter, the default value for these bytes is X'10F2C20080'. If you specify the LONGTERM parameter, the default value is X'05F2C20090'.

If you specify a value with the OPMT parameter and do not specify NOUIO, control is returned to the application program only if an error occurs.

PATH=*pathnum*
specifies the path number for a detail subfile using index support, where *pathnum* is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator. If there is only one index path, do not specify this parameter.

See *TPFDF Database Administration* for more information about path numbers.

RELFC
releases the subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

W-type files are automatically released unless they have been sorted, merged, or checkpointed. In these cases, you must specify the RELFC parameter to release W-type files.

STRIP=*striplen*
discards data from the user data area of an LREC that you do not want to display, where *striplen* is the length of the part of the LREC that you want to discard starting from the beginning of the LREC (or of the user portion of an extended LREC). Specify one of the following:

- A register containing the number of bytes that you want to discard.
- A 2-byte label that contains the number of bytes that you want to discard.
- The number of bytes that you want to discard.

Variable length LRECs contain a 2-byte size field at the front of the user data section. The DBDSP macro automatically discards this field; do not include it in the number of bytes you specify with the STRIP parameter.

Do not use registers R14 or R15 with the STRIP parameter.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

NOORG
specifies that the LRECs are organized in the subfile in no particular order. (NOORG is the default if subfile organization has not been defined in the DBDEF).

WTOPC
specifies the format in which the LRECs are displayed, as follows:

YES
specifies to use the WTOPC format. With the WTOPC format, the maximum length displayed for an LREC is 255 bytes, and the LONGTERM, NOUIO, and OPMT parameters are ignored.

NO
specifies to use OMSG format will be used.

Entry Requirements

None.

Normal Return

SW00RTN is set to zero.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can

code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.

- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- You cannot specify six KEY*n* parameters (KEY1–KEY6) when both of the following conditions are true:
  - The S subparameter is used on all six keys
  - The length of all keys is greater than 1 (either with an explicit length using the L subparameter or an implied length from the field used in the R or T subparameter).

  If you need this type of key definition, you must use a key list.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent macros when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with macros that access or update records without using fullfile processing.
- The subfile you select must contain LRECs with only extended binary code decimal interchange code (EBCDIC) characters that can be displayed (such as, letters, numbers, punctuation, and so on).
- The ECB exits after a successful display unless you specify NOUIO or NOFINAL, or you specify not to exit using OPMT.
- Although the TPFDF product preserves all data levels across TPFDF macro calls, the following exceptions exist when you specify the DBDSP macro:
  - Data level 1 (D1) and data level 3 (D3) are ***not*** data level independent (DLI) if the WTOPC parameter is specified with the NO value (the default), or the YES value is not specified, and DBLCL macro symbol &ACPDBAA is set to zero.
  - Data level 2 (D2) is ***not*** DLI.
- You cannot use this macro with P-type files.
- You can limit the number of output lines displayed by the DBDSP macro by using the #DF_MAX_DSP equate in the ACPDBE macro. See *TPFDF Installation and Customization* for more information about the ACPDBE macro.
- You cannot use this macro in a commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

In the following example, the amount of data to be stripped is equal to the length of the field GR25KEY:

```
DBDSP REF=GR25DF,STRIP==AL2(L'GR25KEY)
```

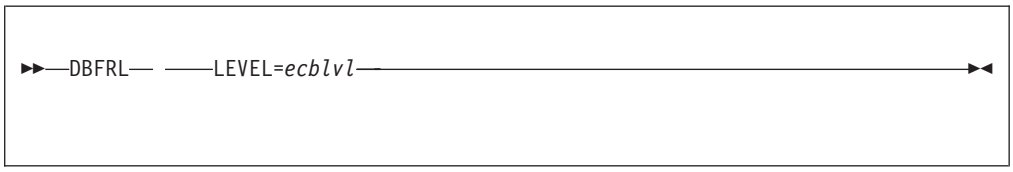
Related Macros

“DBCLS–Close a Subfile” on page 199.

DBFRL–Ensure an ECB Data Level Is Free

Use this macro to free an entry control block (ECB) data level.

Format



LEVEL=*ecblvl*
frees a specific ECB data level, where *ecblvl* is a hexadecimal number in the range X'0'–X'F'. Optionally, you can precede this hexadecimal number with a D to indicate that you are naming a data level.

Entry Requirements

None.

Normal Return

The specified data level is freed.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- If you are processing traditional files with TPFDF files by using the DBOPN macro with the DATA or PARAM parameter specified, you can ensure that a required ECB data level is free by specifying the DBFRL macro before calling a program or function that uses a specific data level. However, the DATA or PARAM parameters are used for migration purposes only.

Examples

Each of the following examples free data level C (also called data level DC) in an ECB. Freeing the data level ensures that the data level is available for use by a traditional TPF or ALCS program and that no TPFDF program can use this level in the ECB.

```
DBFRL LEVEL=C

DBFRL LEVEL=DC
```

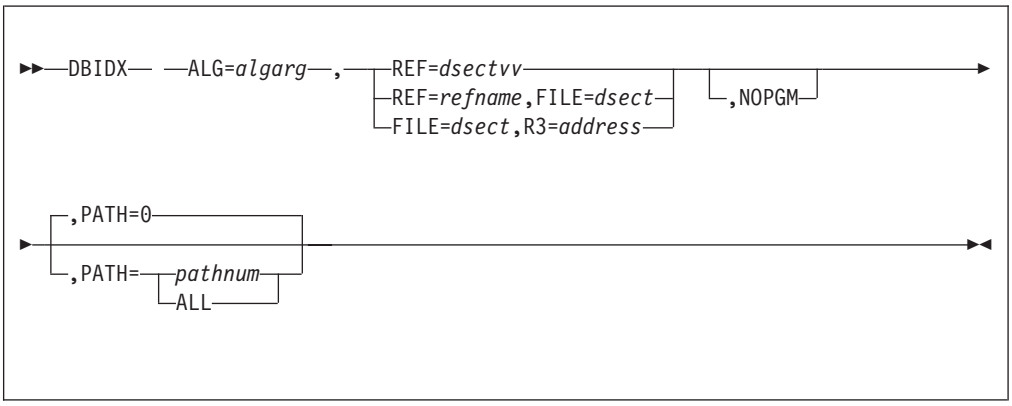
Related Macros

None.

DBIDX–Create an Index Reference

Use this macro to create one or more index references to a subfile identified by an algorithm parameter. You can choose to index one or more paths.

Format



ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

A*I**refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

Entry Requirements

- You must have a detail file available.
- Ensure that the relationship of the index file (or index files, if there are multilevel indexes) to the detail file has been defined in the DBDEF macro by your database administrator.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can

code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.

- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- The TPFDF product determines the amount of data to move in an index LREC by calculating the number of bytes between labels *xxxx*E*yy* and *xxxx*A*yy* in the DSECT for the index file, where *xxxx* is the first 4 characters in the name of the DSECT and *yy* is the primary key.
- Path 0 is the default path. The DBIDX macro creates index references for this path unless you have set one or more different paths using the PATH parameter on the DBIDX or DBOPN macro.
- No actual index structure needs to exist before you index the subfile. All that is needed is an existing index file at the highest level of the index. This must be a fixed file. If there is no existing index structure, the TPFDF product creates the required index structure automatically when you call the DBIDX macro.
- If you index a subfile with the DBIDX macro in an application program, you must also remove the index when needed. (You can do this by using the DBDIX macro.)
- When running in detac mode, if an application program creates a pool file using the DBCRE macro and an index reference using the DBIDX macro, the application program must delete the index reference using DBDIX macro before using the ABORT parameter on the DBCLS macro. If the index reference is not deleted, subsequent recoup processing may identify the index reference as a broken chain.

Examples

The following example creates a new subfile, creates the index reference to that subfile, and adds an LREC to the subfile.

```
DBCRE REF=GR23DF
DBIDX REF=GR23DF,ALG=EBW0044
DBADD REF=GR23DF
```

Related Macros

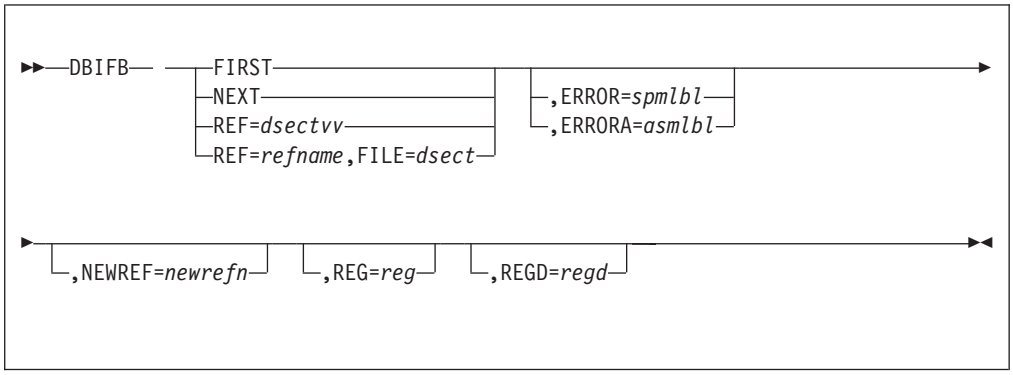
- “DBADD–Add a Logical Record to a Subfile” on page 176
- “DBCRE–Create a Subfile” on page 210
- “DBDIX–Delete Index References to a Subfile” on page 228
- “DBOPN–Open a Subfile” on page 260.

DBIFB–Check a SW00SR Slot

Use this macro to check if a SW00SR slot exists. If the slot exists, the function returns the base address of the SW00SR slot.

You can use this macro to test if a particular subfile is open.

Format



FIRST
inserts the address of the first SW00SR slot with an open subfile in your application program in register 3 (R3). R14 points to the 8-byte location containing the reference name for the file.

NEXT
inserts the address of the next SW00SR slot with an open subfile in your application program in register 3 (R3). R14 points to the 8-byte location containing the reference name for the file.

REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

NEWREF=*newrefn*
changes the reference name of the file specified with the REF parameter, where *newrefn* is one of the following:

- An explicit term that represents the new reference name
- A label in one of the following formats:

  newrefn
  is the label of an 8-byte field that contains the new reference name.

  AI*newrefn*
  is the label of a 4-byte field that contains the storage address of the 8-byte field containing the new reference name.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

Entry Requirements

None.

Normal Return

The base address of the SW00SR slot is returned in register 3 (R3).

Error Return

If the required SW00SR slot is not found, R3 is set to zero. You can test the return value of R3 in the program, or use the ERROR or ERRORA parameters to cause the program to branch if the TPFDF product cannot find a selected SW00SR.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.

Examples

The following example locates the open SW00SR slot of IWA3DF (if it exists) and inserts the address of the current LREC for this SW00SR slot in R2.

DBIFB

```
DBIFB REF=IWA3DF,REG=R2
```

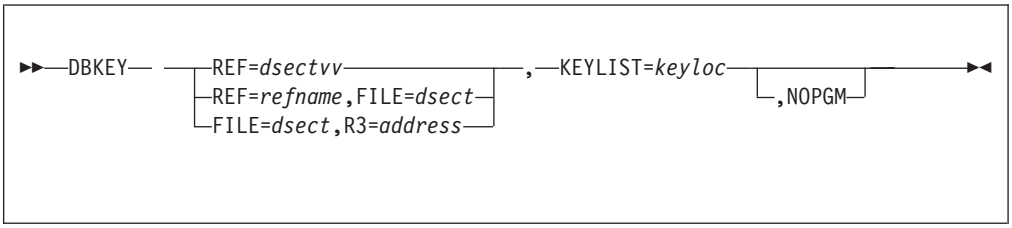
Related Macros

"DBOPN–Open a Subfile" on page 260.

DBKEY–Activate a Key List

Use this macro to activate a key list that is used by subsequent macros that access the specified file. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about keys.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

KEYLIST=*keyloc*
specifies a key list that you want to use with this macro, where *keyloc* is one of the following:

- A register that contains the address of the key list
- A label in one of the following formats:

  keyloc
  is a label indicating the address of the key list.

  AI*keyloc*
  is the label of a 4-byte field that contains the storage address of the key list.

See “Setting Up and Using a Key List” on page 26 for information about how to set up a key list.

DBKEY

NOPGM
specifies not to change the program stamp in a block when filing it.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- The key list can define as many as 180 key parameters or it can indicate that default keys will be used. See “Setting Up and Using a Key List” on page 26 for more information about defining a key list.
- Several macros provide a KEYLIST parameter, which generates an internal DBKEY macro.

Examples

See “Using a Key List with the DBSETK Macro” on page 28 for an example of activating key list with the DBKEY macro.

Related Macros

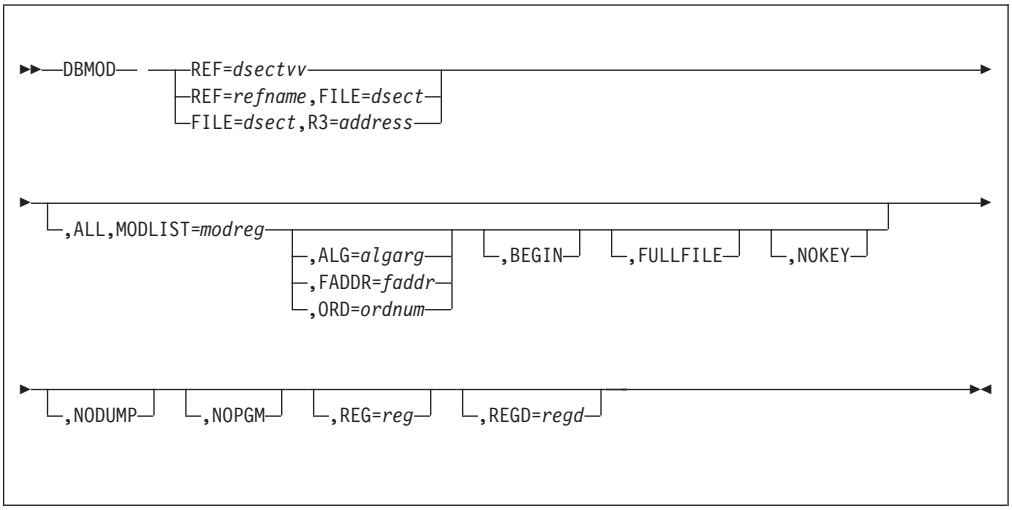
- “DBADD–Add a Logical Record to a Subfile” on page 176
- “DBDEL–Delete One or More Logical Records” on page 214
- “DBDSP–Display Logical Records from a Subfile” on page 231
- “DBMRG–Merge Logical Records from Two Subfiles” on page 254
- “DBOPN–Open a Subfile” on page 260
- “DBRED–Read a Logical Record” on page 272
- “DBSRT–Sort a Subfile” on page 307.

DBMOD–Perform or Indicate Logical Record Modifications

Use this macro to do the following:

- Indicate that the current logical record (LREC) has been modified
- Modify all LRECs in a file or subfile that match previously established keys.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALL
modifies every LREC in the open subfile specified by the REF parameter, beginning with the current LREC. If selection keys are currently active, the DBMOD macro only modifies the LRECs that match these keys.

Note: In an ALCS environment, TPFDF C language support must be enabled if you specify this parameter when KEYCHECK=YES is defined on the DBDEF macro. See *TPFDF Installation and Customization* for more information about enabling TPFDF C language support in an ALCS environment.

MODLIST=*modreg*
specifies the base register of the modification key list, where *modreg* is a register. See “Setting Up and Using a Key List” on page 26 for more information about defining a modification key list.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

A*I**ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

BEGIN
searches from the beginning of the subfile for LRECs to modify.

FULLFILE
modifies LRECs in all subfiles of the file, not just the current subfile.

NOKEY
deactivates any currently active keys.

NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this macro:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the NODUMP parameter is not recommended because it can prevent system errors from being issued that indicate a critical problem.

NOPGM
specifies not to change the program stamp in a block when filing it.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

Entry Requirements

None.

Normal Return

- If you are using the DBMOD macro to indicate that you have modified a record in storage, the address of the current LREC is loaded in the SW00REC field of the SW00SR slot.
- If a global modification is being done, the SW00REC field of the SW00SR slot contains 0 and SW00RTN contains a record not found (#BIT1) or end-of-file (#BIT5) indication. This is a normal return condition.

Error Return

None.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- If you locate an LREC or header in a subfile using the DBRED macro and modify the LREC data using assembler instructions, you must ensure that the changes are recorded on DASD. Use the DBMOD macro without the ALL parameter to do this.

  The DBMOD macro sets an indicator in the block to say that it has been changed. The TPFDF product writes this block to DASD when you close or checkpoint the subfile. You must call the DBMOD macro while the LREC you modified is still current. If you allow the program to read other LRECs in the subfile before you call the DBMOD macro, some modifications to LRECs can be lost.

  Attention: Do not use the DBMOD macro if you have changed:
  - The size of the existing LREC
  - Any key fields
  - Any fields in the LREC that are also used as index key fields.

  Instead, delete the old LREC with a DBDEL macro and add a new LREC with a DBADD macro.
- Use the ALL parameter to perform global modifications. Global modification allows you to update multiple LRECs with a single DBMOD call. You must provide the base register of a modification key list with the MODLIST parameter. The modification key list contains the rules for updating the LRECs. See “Setting Up and Using a Key List” on page 26 for more information about defining a modification key list.
- If you use global modification when KEYCHECK=YES is defined on the DBDEF macro, and any of the fields being modified overlap any default key fields for that primary key in the file, the TPFDF product issues a system error and processing ends. All records that were changed before processing ended remain changed.
- The DBMOD macro does not have a KEYLIST parameter. If necessary, activate the selection key list using the DBKEY macro before calling the DBMOD macro with the ALL parameter to perform global modifications of LRECs.

Examples

The following is an example of how to do a global modification of LRECs from an application written in assembler. In this example, a selection key list is first established using 3 keys as the selection criteria. Then a modification key list is defined. The modification key list indicates that for each LREC that meets the selection criteria, field zzzzFL1 will have the halfword value at EBX020 added to it and the byte at zzzzFL2 will be set to X'00'. Processing will begin with the first LREC in the subfile and end with the last LREC in the subfile.

```
SW01SR REG=R5
LA     R5,EBW000
*
MVC    SW01NKY,=H'3'
DBSETK BASE=R5,KEYNUM=1,DIS=I/zzzzPKY-zzzzREC,LEN=L'zzzzPKY,   *
      CON=#DF_EQ,MSK=I/X'80',ID1=#DF_UP+#DF_CONST
DBSETK BASE=R5,KEYNUM=2,DIS=I/zzzzKY1-zzzzREC,LEN=L'zzzzKY1,   *
      CON=#DF_EQ,SEA=EBX000,ID1=#DF_UP
DBSETK BASE=R5,KEYNUM=3,DIS=I/zzzzKY2-zzzzREC,LEN=L'zzzzKY2,   *
      CON=#DF_NE,SEA=EBX010,ID1=#DF_UP
DBKEY REF=zzzzzz,KEYLIST=EBW000,NOPGM
*
MVC    SW01NKY,=H'2'
DBSETK BASE=R5,KEYNUM=1,DIS=I/zzzzFL1-zzzzREC,LEN=L'zzzzFL1,   *
      SEA=EBX020,ID2=#DF_AH
DBSETK BASE=R5,KEYNUM=2,DIS=I/zzzzFL2-zzzzREC,LEN=L'zzzzFL2,   *
      MSK=I/X'00',ID2=#DF_MVI
*
DBMOD REF=zzzzzz,ALL,BEGIN,MODLIST=R5,REG=R6
```

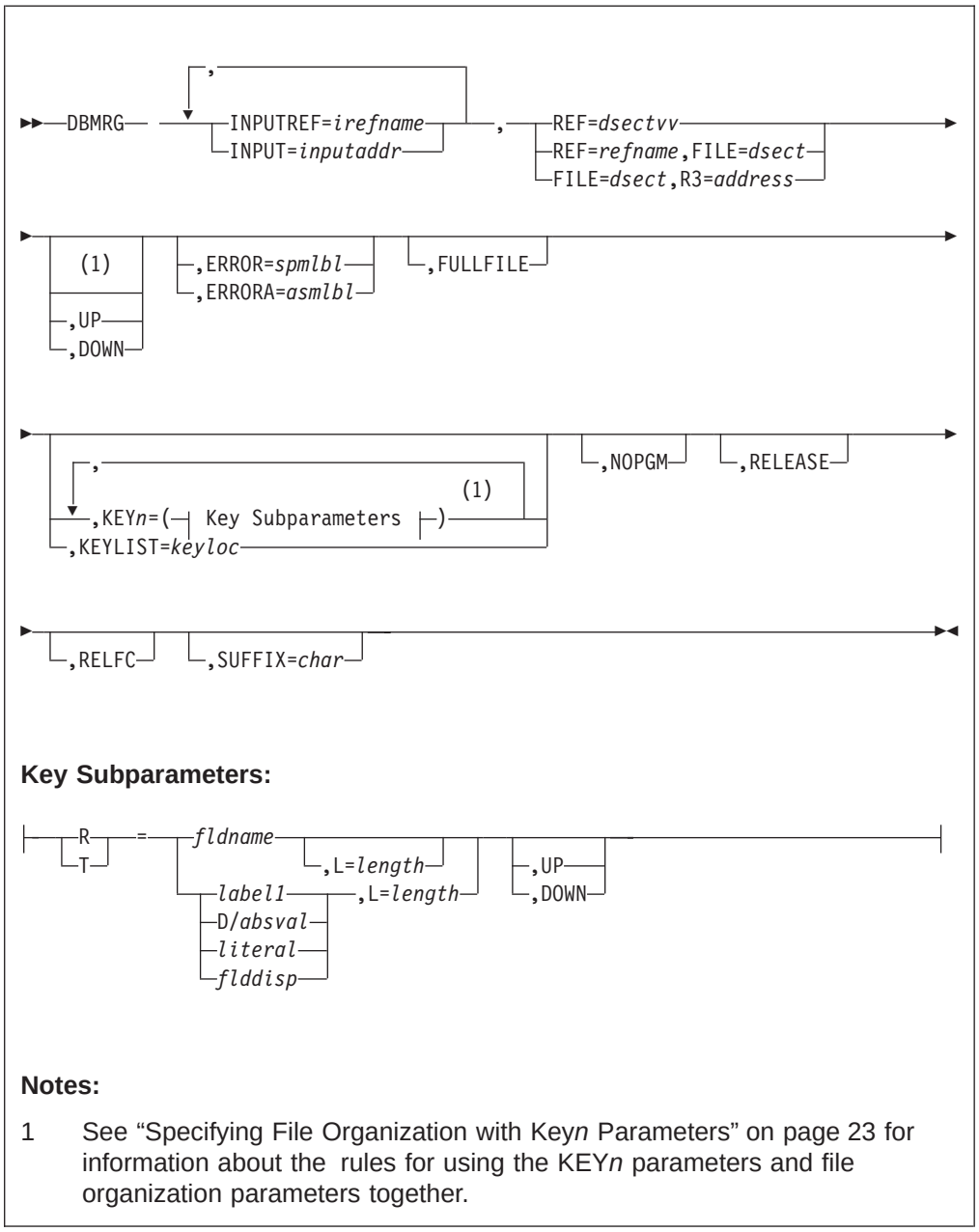
Related Macros

- “DBADD–Add a Logical Record to a Subfile” on page 176
- “DBREP–Replace a Logical Record with Another Logical Record” on page 286.

DBMRG–Merge Logical Records from Two Subfiles

Use this macro to merge two subfiles into one subfile.

Format



Key Subparameters:

Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

INPUTREF=*irefname*

specifies the reference name of the input subfile, where *irefname* is one of the following:

- The DSECT name
- A label that references the DSECT name in one of the following formats:

  irefname

  is the label of an 8-byte field containing the DSECT name.

AI*irefname*
is the label of a 4-byte field that contains the storage address of an 8-byte field containing the DSECT name.

INPUT=*inputaddr*
specifies the base address of the SW00SR slot of the input file.

Note: This parameter is supported for migration purposes only; where possible, always use the INPUTREF parameter to identify the input subfile.

REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FULLFILE
merges LRECs from the entire input file to the output subfile specified with the REF parameter.

KEY*n*

specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See “Specifying File Organization with Key*n* Parameters” on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

R specifies a field in the LREC to be used for placing the LRECs in the output file.

T specifies a field in the subLREC of an extended LREC to be used for placing the LRECs in the output file.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(R=GR00FLD)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(R=EBX010,L==H'4')
```

D*l**absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(R=D/2,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(R=D/GR00NAM-GR00REC,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(R==H'2',L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(R=GR00FLD+2,L==H'4')
```

or

```
... KEY1=(R=GR00FLD+L'GR00FLD,L==H'4')
```

L=*length*
specifies the length of the field to be used in placing the LRECs in the output file, where *length* is one of the following:

- The label of a 2-byte field containing the length of the field
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

KEYLIST=*keyloc*
specifies a key list that you want to use with this macro, where *keyloc* is one of the following:

- A register that contains the address of the key list
- A label in one of the following formats:

  keyloc
  is a label indicating the address of the key list.

  AI*keyloc*
  is the label of a 4-byte field that contains the storage address of the key list.

See “Setting Up and Using a Key List” on page 26 for information about how to set up a key list.

NOPGM
specifies not to change the program stamp in a block when filing it.

RELEASE
releases the SW00SR slot for the input subfile after the macro has completed processing.

RELFC
releases the input subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

You must open both subfiles before calling DBMRG.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.

- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent macros when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with macros that access or update records without using fullfile processing.
- Make sure the input file contains LRECs that are in the same order as the output file. If the input file is not in the same order as the output file, do not merge the files until they are in the same order. Use the DBSRT macro to arrange the input file in the same order as the output file.
- If the output subfile is not a B+Tree subfile, you ***must*** specify keys using KEY*n* parameters or the KEYLIST parameter when you call this macro. The keys specify how the TPFDF product sorts the LRECs in the output subfile.

  The KEY*n* and KEYLIST parameters are ignored for B+Tree files. The output file is organized according default keys defined on the DBDEF macro for the file. See *TPFDF Database Administration* for more information about default keys.
- The output file cannot be in detac mode when you use the DBMRG macro.
- You cannot issue additional TPFDF macros to the input file until the file is closed if the following conditions are true:
  – You specify the FULLFILE parameter
  – You do not specify the RELEASE parameter
  – The end-of-file indicator is set.

  However, you can specify the REUSE parameter on the DBCLS macro. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- When this macro has completed processing, the output subfile is left open and must be closed using the DBCLS macro before the ECB exits. If you specify the RELEASE parameter, this macro closes the input subfile.
- You cannot use this macro with P-type files.
- Figure 20 shows how the DBMRG macro merges LRECs from two subfiles.

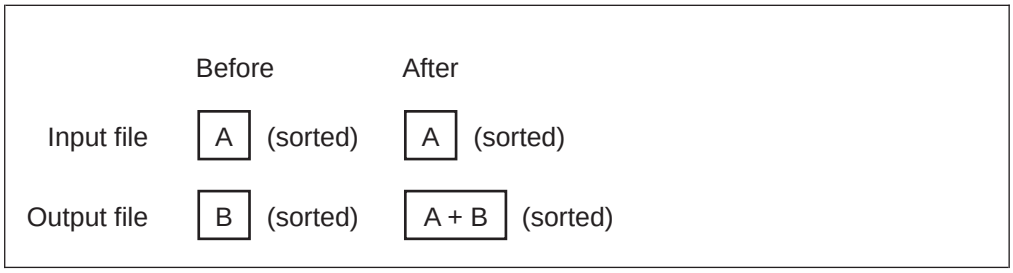


Figure 20. Merging LRECs from Two Subfiles. The input file is defined by the INPUTREF parameter and the output file is defined by the REF parameter.

  This macro does not modify the input subfile that you specify with the INPUTREF parameter. The TPFDF product leaves the ECB data level for the input subfile free after use.
- If you use the DBMRG macro in a commit scope, the files or subfiles that are being merged must be opened in the same commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

The following example merges files GR26DF and GR25DF.

```
DBMRG REF=GR26DF,INPUTREF=GR25DF,                *
      KEY1=(R=GR25KEY,L==AL2(L'GR25KEY))         *
```

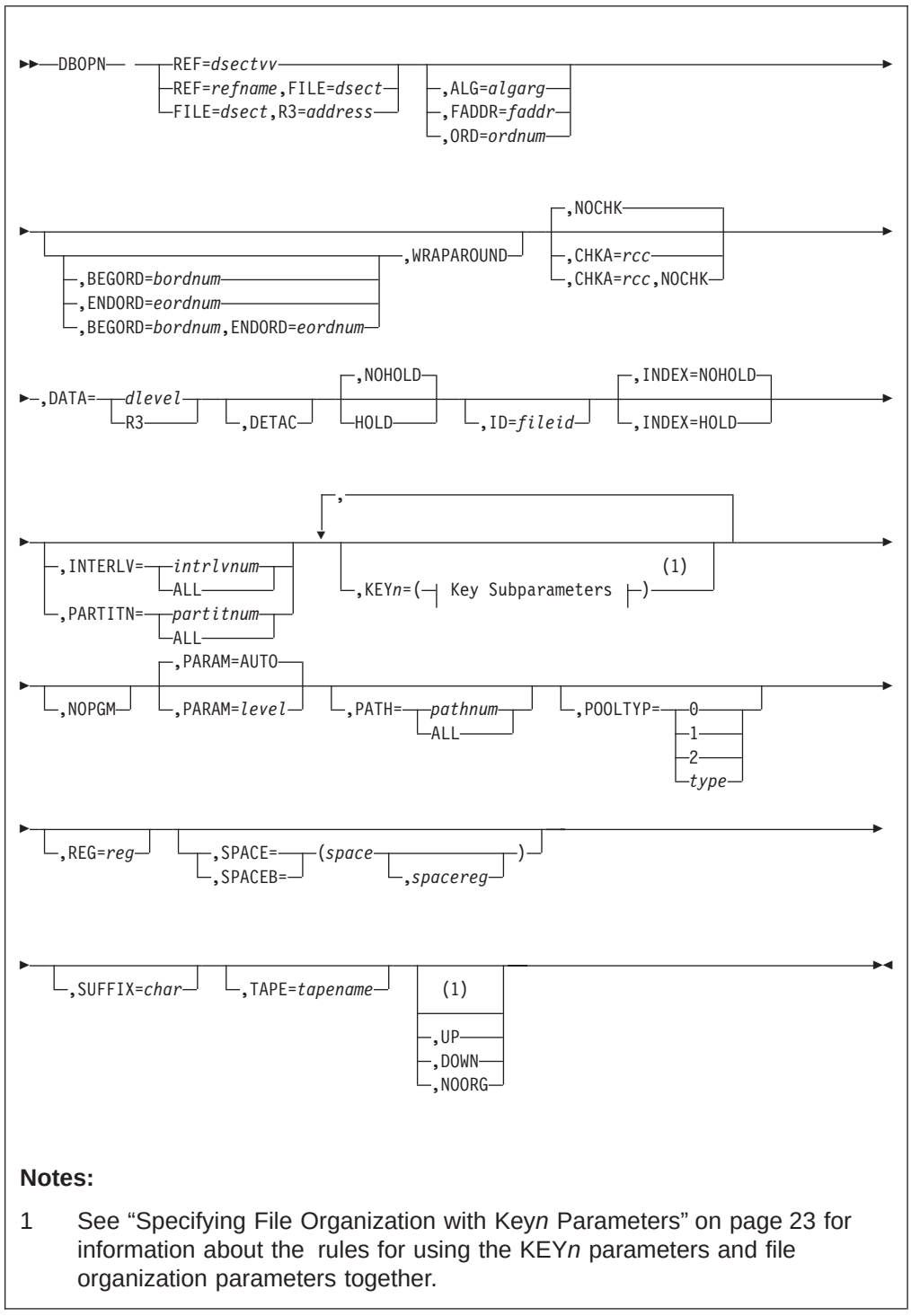
Related Macros

"DBSRT–Sort a Subfile" on page 307.

DBOPN–Open a Subfile

Use this macro to open a subfile. This is the first TPFDF macro you use with any file. The first DBOPN macro creates a database interface block (DBIFB) and reserves a SW00SR slot in the DBIFB. The SW00SR slot contains control information about the subfile. See *TPFDF General Information* for more information about SW00SR and this control information.

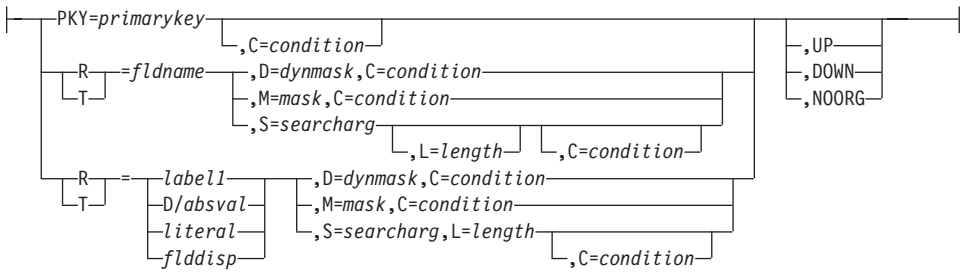
Format



Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

Key Subparameters:



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that will contain the address of the SW00SR slot on return from the macro.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument

- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

BEGORD=*bordnum*
overrides the default starting ordinal number for use in a subsequent DBRED macro or other macro statement, where *bordnum* is a label pointing to a 4-byte field containing the required ordinal number.

ENDORD=*eordnum*
overrides the default ending ordinal number for use in a subsequent DBRED macro or other macro statement, where *bordnum* is a label pointing to a 4-byte field containing the required ordinal number.

WRAPAROUND
sets a *wraparound* bit. If you use other TPFDF macros with the FULLFILE parameter with this file, the TPFDF product processes the subfiles in order, returning to the first subfile from the last subfile.

For example, consider a file that contains 5 subfiles. If you specify BEGORD=3 and the WRAPAROUND parameter with the DBOPN macro, and then code a DBRED macro with the FULLFILE parameter specified, the TPFDF product processes the subfiles in order 3, 4, 0, 1, 2, then indicates end of file.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

DATA
specifies the data level for the data block. Specify one of the following:

dlevel
is the data level.

R3
is for internal TPFDF use only.

Note: Do not use this parameter; it is provided only for migration purposes. In addition, do ***not*** specify the DATA parameter if you have specified the REF parameter in your application program.

DETAC
opens the subfile in detac mode. When the subfile is in detac mode, all modified blocks are saved in main storage. No blocks are written to DASD until you checkpoint the subfile (using the DBCKP macro) or close the subfile (using the DBCLS macro). Access to LRECs in blocks in main storage can be very fast.

If you do not want to keep any modifications that you made to the subfile opened with the DETAC parameter, you can use the ABORT parameter with the DBCLS macro. This closes the file without saving any modifications to disk.

Each subfile that you open with the DETAC parameter uses up some main storage, so avoid using this parameter unnecessarily.

Notes:

1. When you are using fullfile processing, each subfile in turn is put in detac mode, not the entire file.
2. The TPF system and the ALCS environment issues a 000010 system error if an application program does not give up control in the time allotted by the application time-out counter. When processing in detac mode, a TPFDF application program can require more than the allotted time on a database with a large data structure. To prevent the 000010 system error, you can change the setting of the &TPFDBDV symbol in the DBLCL macro.

   See *TPFDF Installation and Customization* for more information about the &TPFDBDV symbol and the DBLCL macro.

HOLD
potentially holds the subfile that you are accessing and prevents two or more application programs from modifying the subfile at the same time. Holding occurs on the following TPFDF call that accesses the subfile if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the subfile will not occur until the subfile is no longer held. If more than one application can update the same subfile, or when the file is processed in fullfile mode, you must specify this parameter to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP=2 parameter in the DBDEF macro, affect hold processing.

NOHOLD
does not hold the subfile that you are accessing. You can specify this parameter when you open a subfile and are not going to make modifications to it.

ID=*fileid*
specifies the ID of the file to be opened, where *fileid* is a halfword containing the file ID. This parameter is for use by the database administrator only; for example, in ZUDFM utilities for performing centralized database routines.

INDEX=HOLD
potentially holds any index files that reference the subfiles you are accessing and prevents two or more application programs from modifying the index files at the same time. Holding occurs on the following TPFDF call that accesses the subfile if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the index file will not occur until the index file is no longer held. If more than one application can update the same index file, or the file is processed in fullfile mode, you must specify this parameter to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT marco, or the OP2= parameter in the DBDEF macro, affect hold processing.

INDEX=NOHOLD
does not hold the index files that reference the subfiles you are accessing.

INTERLV
specifies the interleave that you want to use. Specify one of the following:

interlvnum
is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

ALL
specifies all interleaves. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different interleave.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN
specifies the partition that you want to use. Specify one of the following:

partitnum
is one of the following:
- A register that contains the address of partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

ALL
specifies all partitions. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different partition.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm computes the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

KEY*n*
specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See "Specifying File Organization with Key*n* Parameters" on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

PKY=*primarykey*
specifies a value that will be compared against the primary key of an LREC, where *primarykey* is a 1-byte immediate value; for example:

```
... KEY1=(PKY=#RR00K80)
```

This has the same effect as:

```
... KEY1=(R=RR00KEY,S=#RR00K80)
```

R specifies a field in the LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

T specifies a field in the subLREC of an extended LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,S=EBW000)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=EBX010,S=EBW000,L==H'4')
```

D*absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/2,S=EBW000,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/GR00NAM-GR00REC,S=EBW000,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R==H'2',S=EBW000,L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+2,S=EBW000,L==H'4')
```

or

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+L'GR00FLD,S=EBW000,L==H'4')
```

C=*condition*
specifies the condition to be used when comparing fields in the logical record (specified with the R subparameter) with the search argument (specified with the S or PKY subparameter) or with the bit mask (specified with the M or D subparameter).

If you specify the S or PKY subparameter, use one of the following values:

Value	Condition
EQ	Equal (this is the default)
E	Equal
NE	Not equal
GE	Greater than or equal
LE	Less than or equal
GT	Greater than
LT	Less than
H	High
L	Low
NH	Not high
NL	Not low.

If you specify the M or D subparameter, use one of the following values:

Value	Condition
Z	Zeros
O	Ones
M	Mixed
NZ	Not zeros
NO	Not ones
NM	Not mixed.

D=*dynmask*
specifies the label of a 1-byte field containing a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,D=EBW000,C=Z)
```

M=*mask*
specifies a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,M=X'80',C=Z)
```

S=*searcharg*
specifies the search argument to be compared with the LREC field specified with the R or T subparameter, where *searcharg* is one of the following:

- A register that contains the address of the search argument
- A literal that represents the search argument
- A label in one of the following formats:

  searcharg
  is the label of the search argument.

  AI*searcharg*
  is the label of a 4-byte field that contains the storage address of the search argument.

P/*searcharg*
is the label of a field that contains the search argument in packed decimal format.

If you specify P/*searcharg* or a literal in the form of =P"...", the LREC field and search argument are compared as decimal numbers in packed format. Otherwise, the LREC field and search argument are compared as character data.

L=*length*
specifies the length of the search argument, where *length* is one of the following:

- The address of a 2-byte field containing the length of the search argument
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

NOORG
specifies that the key field is in no particular order in the subfile.

NOPGM
specifies not to change the program stamp in a block when filing it.

PARAM
specifies the entry control block (ECB) data level for the SW00SR space to be allocated. Specify one of the following:

level
is the data level.

AUTO
specifies automatic SW00SR allocation. Use this value if you specified the DATA parameter.

Note: Do not use this parameter; it is provided only for migration purposes.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

POOLTYP
overrides the pool type defined by the database administrator, where:

0 uses the pool type defined by the PF0 parameter of the DBDEF macro.

1 uses the pool type defined by the PF1 parameter of the DBDEF macro.

2 uses the pool type defined by the PF2 parameter of the DBDEF macro.

type
is the label of a 1-byte field that contains a 0, 1, or 2 to specify to pool type.

Use the POOLTYP parameter as directed by the database administrator.

REG=*register*
generates the DSECT macro specified with the REF or FILE parameter and generates a USING statement to provide addressability to the DSECT, where *register* is the register that will be used on the USING statement. If you specify the SUFFIX parameter as well, the DBOPN macro generates the DSECT using the specified suffix.

If you do not specify the REG parameter and if the application program needs access to the DSECT labels, you must code the DSECT in the application source code directly.

SPACE
provides work space when you open a subfile and initializes the work space to X'00'

SPACEB
provides work space when you open a subfile and initializes the work space to X'40'

space
is the number of bytes of space you want, which can be a maximum of 3952 bytes. Specify one of the following:

- An absolute value. For example,
  ```
  DBOPN REF=GR36DF,SPACE=(400,R5)
  ```
- The length of a label. For example,
  ```
  DBOPN REF=GR36DF,SPACE=(L'GR36REC,R5)
  ```
  This provides space that is the same length as the field.
- The label of a field. For example,
  ```
  DBOPN REF=GR36DF,SPACE=(EBW002,R5)
  ```
  The space is determined by the 2 bytes of data beginning at the specified label.

spacereg
is the register in which you want the base address of the work space loaded.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

TAPE=*tapename*
specifies the tape or sequential data set to be used when creating overflow blocks, where *tapename* is a 3-character symbolic tape name. When you specify this parameter, if you are adding LRECs and the prime block overflows, the TPFDF product copies the current prime block to the specified tape or sequential data set). The TPFDF product then initializes the original prime block by inserting a new standard TPFDF header and adds the new LREC to this prime block.

DBOPN

This parameter is useful when you are logging data to a real-time tape.

Note: B+Tree files cannot be opened using the TAPE parameter.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

NOORG
specifies that the LRECs are organized in the subfile in no particular order. (NOORG is the default if subfile organization has not been defined in the DBDEF).

Entry Requirements

Ensure that the subfile you open was previously defined in a DSECT macro and in a DBDEF macro by your database administrator.

Normal Return

The address of the SW00SR slot.

Error Return

The error indicators in the SW00RTN field of the SW00SR slot have no meaning for this macro.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- You cannot specify six KEY*n* parameters (KEY1–KEY6) when both of the following conditions are true:
  - The S subparameter is used on all six keys
  - The length of all keys is greater than 1 (either with an explicit length using the L subparameter or an implied length from the field used in the R or T subparameter).

  If you need this type of key definition, you must use a key list.
- The following rules determine the value of the record code check (RCC) value used when the TPFDF product creates a new subfile:
  - If you do not specify the NOCHK parameter, the TPFDF product creates new subfiles with a random RCC value.

  - If you specify the NOCHK parameter without the CHKA parameter, the TPFDF product creates new subfiles without an RCC value.
  - If you specify both the NOCHK and CHKA parameters, the TPFDF product creates new subfiles with the RCC value specified with the CHKA parameter.
- When you use the REF parameter to open a file, the TPFDF product automatically allocates a SW00SR slot and loads the address in register 3 (R3). The DSECT name you used is put in the SW00SR slot as the *file reference*. Use the same REF parameter value with other macros to identify the file.

  If you specify the FILE parameter alone, this implies that R3 was set up by the application program to point to the correct SW00SR slot.
- You can open selected subfiles of a particular file simultaneously. To do this, use the same 6-character DSECT name for all subfiles, but follow it with different appendixes (for example, 01, 02, 03, and so on) to identify each subfile.

  Keep in mind that each subfile has a different SW00SR. When you access a subfile you must use the same 8-character reference name that you used to open the subfile.
- If you specify the BEGORD and ENDORD parameters, subsequent macros (such as a DBRED macro with the FULLFILE parameter specified) process the file only between the defined begin and end ordinals.
- You cannot use this macro with a T-type file because a T-type file is a temporary logical record (LREC) stored in a W-type file and is not defined in the database definition (DBDEF) macro. See *TPFDF Database Administration* for more information about T-type files.
- If you use this macro with a W-type file, the HOLD parameter is the default. See *TPFDF Database Administration* for more information about W-type files.
- It is not necessary to specify data level independence (DLI) with this macro. The TPFDF product preserves all data levels holding blocks before a macro or function call. However, if you use the DBOPN macro with the DATA and PARAM parameters specified, the data is returned to the data level specified.

  Note: The DATA and PARAM parameters are provided for migration purposes only.

  See “Data Level Usage” on page 3 for more information about DLI.

Examples

The following example opens two different subfiles and then later accesses one of them.

```
DBOPN REF=GR00SR01,ALG==C'A'
DBOPN REF=GR00SR02,ALG==C'B'
  ⋮
DBRED REF=GR00SR01
```

Related Macros

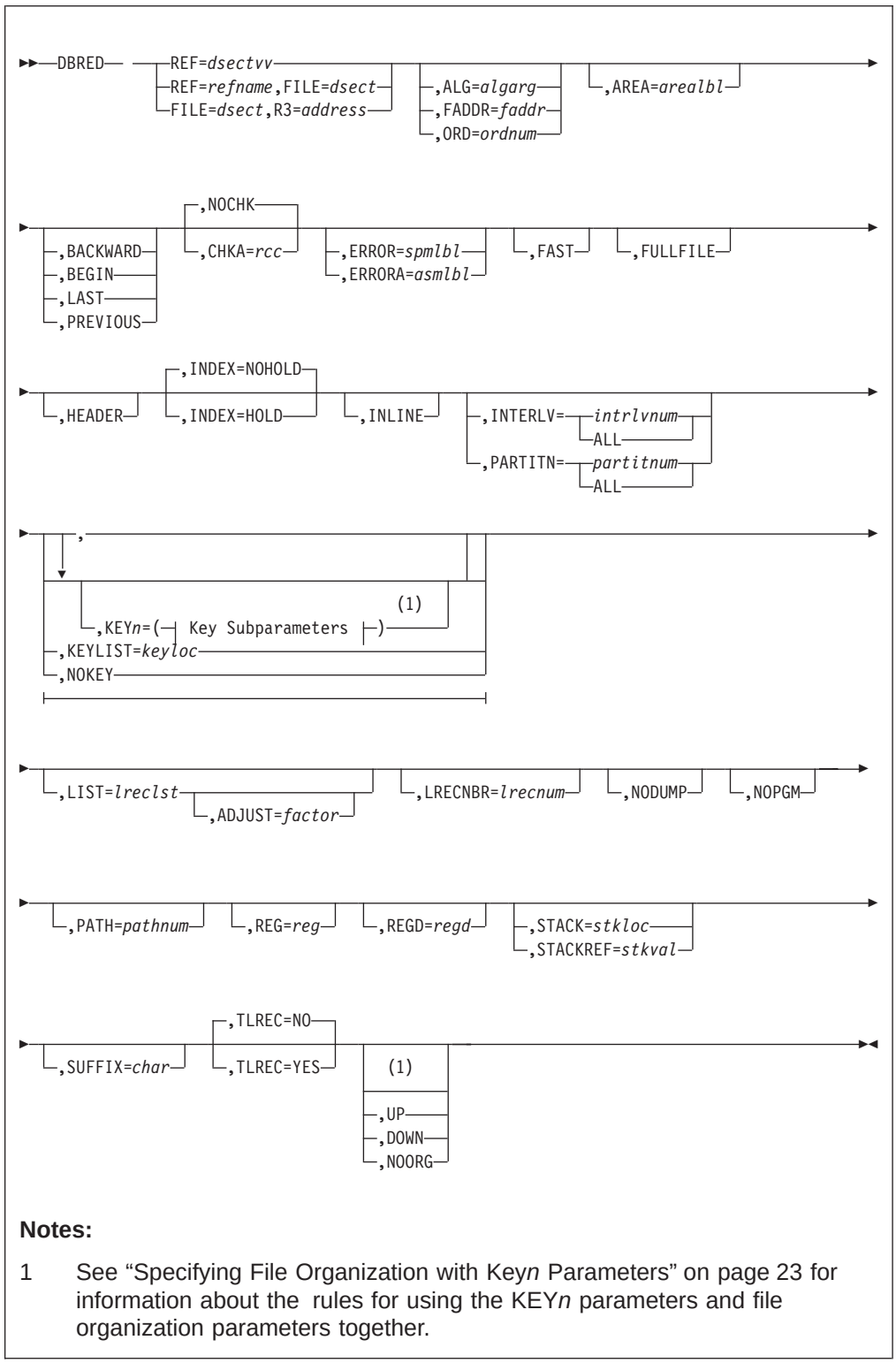
“DBCLS–Close a Subfile” on page 199.

DBRED–Read a Logical Record

Use this macro to read a logical record (LREC) or block header and get the address where the record is stored. You can read the next LREC in sequence or specify details of the LREC you require.

You can also use this macro to read a sequence of LRECs. In this case, you perform a sequence of DBRED macro calls and get a different LREC each time.

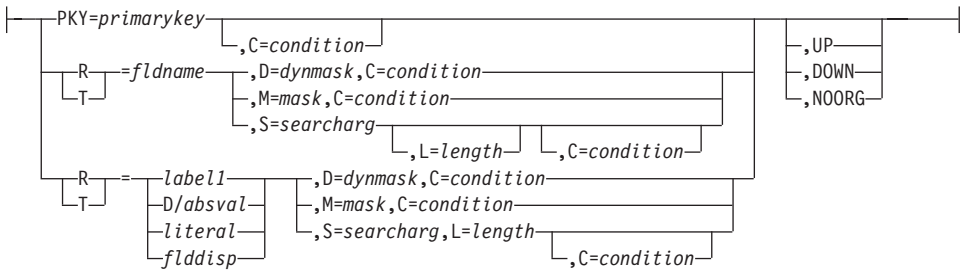
Format



Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

Key Subparameters:



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

A*I**refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument

- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

AREA=*arealbl*
specifies an area to which the TPFDF product copies user data from the index LREC referencing the detail subfile you are accessing, where *arealbl* is one of the following:

arealbl
is the label of a field to which the user data will be copied.

AI*arealbl*
is the label of a 4-byte field that contains the storage address of the field to which the user data will be copied.

The user data is defined in the DBDEF macro of the detail subfile. See *TPFDF Database Administration* for more information about defining this user data.

BACKWARD
reads through a subfile backwards. The DBRED macro reads the LREC immediately preceding the current one.

Notes:

1. You cannot use the BACKWARD parameter with KEY*n* or LRECNBR parameters.
2. If you use the BACKWARD parameter with the DBRED macro and also use the DBRET macro, you must specify the STACK or STACKREF parameter with the DBRET macro. See “DBRET–Retain a Logical Record Position” on page 290 for more information about these parameters and the DBRET macro.
3. If you use the BACKWARD parameter, the default and recommended setting for symbol &DB013A in the DBLCL macro is 0. This setting requires files to use full backward chaining (bit 0 of &SW00OP1 is set) to read backward. See *TPFDF Database Administration* for more information about defining full backward chaining. See *TPFDF Installation and Customization* for more information about the DBLCL macro.

Exception: If the file uses add current processing (bit 2 of &SW00OP1 is set) with no chains (&SW00NOC = 0), you can code the DBRED macro with the BACKWARD parameter specified regardless of how bit 0 of &SW00OP1 is set.

BEGIN
reads the subfile from the beginning rather than from the current LREC. This ensures that the DBRED macro locates the first LREC in the file that matches any search arguments you have specified.

LAST
reads the last LREC in a subfile. If you use the LAST parameter together with KEY*n* parameters, the DBRED macro supplies the last LREC that matches the search arguments.

PREVIOUS
retrieves an LREC that you saved using the DBRET macro.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FAST
used for migration purposes only; use the INLINE or NOKEY parameter instead. If you specify this parameter, the NOKEY parameter is implemented; that is, keys that are currently active are deactivated.

FULLFILE
reads an LREC (specified using the ALG and KEY*n* parameters) from any of

the subfiles in the file referenced by the REF parameter. If you do not specify the FULLFILE parameter, you can only read an LREC in the current subfile.

Note: If the BEGORD and ENDORD parameter were defined on the DBOPN macro, the DBRED macro with the FULLFILE parameter only accesses the subfiles between the specified ordinals.

HEADER
locates the subfile header in the prime block and returns the address in field SW00REC rather than the address of an LREC. The address of the prime block is also returned in field SW00PCA..

If you specify the HEADER and FULLFILE parameters on an open subfile, the DBRED macro retrieves the header of the next subfile.

INDEX=HOLD
potentially holds any index files that reference the subfiles you are accessing and prevents two or more application programs from modifying the index files at the same time. Holding occurs if bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, have been set appropriately. Subsequent TPFDF calls by other ECBs to modify the index file will not occur until the index file is no longer held. If more than one application can update the same index file, or the file is processed in fullfile mode, you must specify this parameter to ensure the updates are synchronized.

See *TPFDF Database Administration* for information about how bits 4 and 5 in the &SW00OP2 global set symbol in the DSECT macro, or the OP2= parameter in the DBDEF macro, affect hold processing.

INLINE
provides inline processing for this macro. You cannot use this parameter with the KEY*n* or LRECNBR parameters. Any keys that are active from previous TPFDF macros are deactivated.

Use this parameter when you want to read each LREC in a subfile sequentially.

Notes:

1. After a DBRED macro is issued with the INLINE parameter specified, you cannot be certain that the TPFDF product will check the keys if you issue a DBMOD macro when KEYCHECK=YES is defined on the DBDEF macro.
2. You cannot use this parameter with P-type files.

INTERLV
specifies the interleave that you want to use. Specify one of the following:

interlvnum
is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

ALL
specifies all interleaves. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different interleave.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN
specifies the partition that you want to use. Specify one of the following:

partitnum
is one of the following:
- A register that contains the address of partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

ALL
specifies all partitions. Use this value when you use fullfile processing to ensure that you do not miss an LREC located in a different partition.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm computes the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

KEY*n*
specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See “Specifying File Organization with Key*n* Parameters” on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

PKY=*primarykey*
specifies a value that will be compared against the primary key of an LREC, where *primarykey* is a 1-byte immediate value; for example:

```
... KEY1=(PKY=#RR00K80)
```

This has the same effect as:

```
... KEY1=(R=RR00KEY,S=#RR00K80)
```

R specifies a field in the LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

T specifies a field in the subLREC of an extended LREC to be compared with the search argument specified with the S subparameter or to be tested against the mask specified with the M or D subparameter.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,S=EBW000)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=EBX010,S=EBW000,L==H'4')
```

D*l**absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/2,S=EBW000,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=D/GR00NAM-GR00REC,S=EBW000,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R==H'2',S=EBW000,L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+2,S=EBW000,L==H'4')
```

or

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD+L'GR00FLD,S=EBW000,L==H'4')
```

C=*condition*
specifies the condition to be used when comparing fields in the logical record (specified with the R subparameter) with the search argument (specified with the S or PKY subparameter) or with the bit mask (specified with the M or D subparameter).

If you specify the S or PKY subparameter, use one of the following values:

Value	Condition
EQ	Equal (this is the default)
E	Equal
NE	Not equal
GE	Greater than or equal
LE	Less than or equal
GT	Greater than
LT	Less than
H	High
L	Low
NH	Not high
NL	Not low.

If you specify the M or D subparameter, use one of the following values:

Value	Condition
Z	Zeros
O	Ones
M	Mixed
NZ	Not zeros
NO	Not ones
NM	Not mixed.

D=*dynmask*
specifies the label of a 1-byte field containing a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,D=EBW000,C=Z)
```

M=*mask*
specifies a mask to be tested against the LREC field specified with the R or T subparameter; for example:

```
... KEY1=(PKY=#GR00K80),KEY2=(R=GR00FLD,M=X'80',C=Z)
```

S=*searcharg*
specifies the search argument to be compared with the LREC field specified with the R or T subparameter, where *searcharg* is one of the following:

- A register that contains the address of the search argument
- A literal that represents the search argument

- A label in one of the following formats:

  searcharg
  is the label of the search argument.

  A*l**searcharg***
  is the label of a 4-byte field that contains the storage address of the search argument.

  P*l**searcharg***
  is the label of a field that contains the search argument in packed decimal format.

  If you specify P/*searcharg* or a literal in the form of =P"...", the LREC field and search argument are compared as decimal numbers in packed format. Otherwise, the LREC field and search argument are compared as character data.

L=*length*
specifies the length of the search argument, where *length* is one of the following:

- The address of a 2-byte field containing the length of the search argument
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

NOORG
specifies that the key field is in no particular order in the subfile.

KEYLIST=*keyloc*
specifies a key list that you want to use with this macro, where *keyloc* is one of the following:

- A register that contains the address of the key list
- A label in one of the following formats:

  keyloc
  is a label indicating the address of the key list.

  A*l**keyloc***
  is the label of a 4-byte field that contains the storage address of the key list.

See “Setting Up and Using a Key List” on page 26 for information about how to set up a key list.

NOKEY
deactivates any currently active keys.

LIST=*lreclst*
reads a list of LRECs, where *lreclst* is a register that contains the address of a list of LREC numbers. The TPFDF product reads these LRECs sequentially when you make a series of DBRED macro calls. The list contains one or more

LREC sequence numbers separated by a slash (/). You can also specify a range of sequence numbers by separating the beginning and end of the range by a hyphen (-). You can use LAST to mean the last LREC of the subfile and ALL to mean all the remaining LRECs. You can also end the list with a nonnumeric character.

Notes:

1. The ranges must be in ascending order; if one is found out of order, that range and all subsequent ranges are ignored.

   For example, if there are 41 LRECs in a subfile, the following lists all have the same effect:

   ```
   20/31/32/33/37/38/39/40/41
   20/31/32/33/37-41
   20/31-33/37-LAST
   20/31-33/37/ALL
   ```

2. You ***cannot*** specify the number zero in the list of LREC numbers, even if you specify the ADJUST parameter with a value that would adjust the number zero to a valid LREC number.

ADJUST=*factor*

specifies an adjustment factor for the values in the list, where *factor* is a register or the label of a field that contains a positive or negative adjustment factor. For example, if you set the adjustment factor to 1, and the list contains LRECs 1/3/5, the DBRED macro reads LRECs 2/4/6.

LRECNBR=*lrecnum*

specifies the sequence number of an LREC that you want to access, where *lrecnum* is one of the following:

- A register that contains the address of the LREC number
- An immediate value that represents the LREC number
- A label in one of the following formats:

  lrecnum
  : is the label of a 4-byte field that contains the LREC number.

  A*lrecnum*
  : is the label of a 4-byte field that contains the storage address containing the LREC number.

Notes:

1. If you use the #TPFDB0D algorithm, you must specify this parameter.
2. LRECs are numbered in increasing order from the start of the subfile (the first LREC in the prime block has sequence number 1).
3. If you specify the LRECNBR parameter with KEY*n* parameters, only those LRECs that match the key conditions are included in the sequence numbering; LRECs that do not match are ignored.

NODUMP

specifies that you do not want the TPFDF product to issue any of the following system errors while processing this macro:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the NODUMP parameter is not recommended because it can prevent system errors from being issued that indicate a critical problem.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH=*pathnum*
specifies the path number for a detail subfile using index support, where *pathnum* is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator. If there is only one index path, do not specify this parameter.

See *TPFDF Database Administration* for more information about path numbers.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). If the HEADER parameter is specified, the address of the prime block (contained in SW00SR field SW00PCA) is returned in the register. You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

STACK=*stkloc*
reads an LREC that you saved using the DBRET macro, where *stkloc* is the location of a 10-byte field that contains the details about the LREC. Specify the same value you used with the STACK parameter on the previous DBRET macro call. See “DBRET–Retain a Logical Record Position” on page 290 for more information about saving LRECs.

Note: Where possible, use the STACKREF parameter.

STACKREF=*stkval*
reads an LREC that you saved using the DBRET macro, where *stkval* is a value assigned to the retained LREC. Specify the same value you used with the STACKREF parameter on the previous DBRET macro call. See “DBRET–Retain a Logical Record Position” on page 290 for more information about saving LRECs.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

TLREC
specifies one of the following:

YES
includes technical LRECs (TLREC) when searching for LRECs to be read.

NO
does not include TLRECs when searching for the LRECs to be read.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

NOORG
specifies that the LRECs are organized in the subfile in no particular order. (NOORG is the default if subfile organization has not been defined in the DBDEF).

Entry Requirements

If you specify the PREVIOUS, STACK, or STACKREF parameter, you must first open the subfile using the DETAC or HOLD parameters.

Normal Return

If the specified LREC is located, the address of that LREC (the *current LREC*) is loaded into the SW00REC field of the SW00SR slot. If you specify the REG parameter, the address is also loaded into the register that you specify. For P-type files, the current LREC is the address of the block that contains the record.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.
- If you use the LIST parameter, you must check bit 1 of the SW00RTN field. If bit 1 in SW00RTN is set, bit 0 in SW00RT2 indicates that the TPFDF product could not process the list successfully.
- If you use the FULLFILE parameter, the TPFDF product does not set any error indicators in SW00RTN except bit 5. It sets this bit when it has reached the end of the complete file. The TPFDF product provides an error count in SW00RT1. You can check this to determine how many errors have occurred since DBOPN when the FULLFILE parameter has been used.

  If you use the FULLFILE parameter, and the end-of-file indicator is set, you cannot issue additional TPFDF macros until the file is closed. However, you can specify the REUSE parameter on the DBCLS macro. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- You cannot specify six KEY*n* parameters (KEY1–KEY6) when both of the following conditions are true:
  – The S subparameter is used on all six keys

  - The length of all keys is greater than 1 (either with an explicit length using the L subparameter or an implied length from the field used in the R or T subparameter).

  If you need this type of key definition, you must use a key list.
- Any active keys are ignored when you use the #TPFDB0D algorithm.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent macros when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with macros that access or update records without using fullfile processing.

  Note: The header parameter can be used with the FULLFILE parameter to move from subfile to subfile, which can then be accessed using the DBRED macros that do not use the FULLFILE parameter. For example:

```
#DO INF
   DBRED REF=GR22DF,FULLFILE,HEADER   * Start of next subfile *
#DOEX DBEOF,YES                       * Exit if end of file *
   DBRED KEYLIST=EBW0000              * Access record in subfile *
#EDO
```

- If you do not specify any search parameters with the DBRED macro (and none are still in effect from the DBOPN macro), the TPFDF product locates the next LREC in sequence in the subfile.
- Do not use the FAST parameter with P-type files.
- To ensure that an LREC is retrieved accurately when you use the DBRED macro, do not use the DBRET macro with the STACK and STACKREF parameters specified on the same open file. If you do, the wrong LREC could be retrieved. You must close and reopen the file each time you alternate between specifying the STACK and STACKREF parameters.

Examples

- The following example shows how to read an LREC from a detail file using index support. First, open the detail file normally, using the REF parameter. Then, use the DBRED macro to read the LREC by providing the index key string of an index LREC as an algorithm argument in the ALG parameter.

```
DBOPN REF=GR44DF
DBRED REF=GR44DF,ALG==C'ALDER'
```

- The following example shows how to read a subfile using read-only default keys in an assembler application. The TPFDF product will search through LRECs in the subfile until it finds a record that matches all of the criteria of the default keys references in the key list. In this example, read-only key X'06', which uses three key fields, is specified. See “Using Default-Key Key Lists” on page 31 for more information about using default keys on read operations.

```
SW01SR  REG=R5                    ADDRESSABILITY TO KEY LIST
GR95SR  REG=R4                    PROTOTYPE LREC STRUCTURE
LA    R4,EBW004                   BASE OF PROTOTYPE LREC
MVI   GR95KEY,X'80'               SEARCH FOR PKY X'80'
MVC   GR95NAM(5),=C'SMITH'        SEARCH FOR NAME OF SMITH
MVC   GR95CTY(8),=C'NEW YORK'     SEARCH FOR CITY OF NEW YORK
LA    R5,EBX000                   LOAD BASE OF DBKEY TABLE
XC    EBX000(L'SW01NKY+L'SW01KIT),EBX000   CLEAR KEYLIST AREA
MVC   SW01NKY,=H'1'               SET UP 1 KEY (READ-ONLY)
DBSETK BASE=R5,KEYNUM=1,MSK=I/X'06',SEA=EBW004,ID1=#DF_KEYS
DBRED REF=IR73DF,KEYLIST=EBX000   ACTIVATE KEY LIST
```

- The following example shows how to read a subfile using OR Boolean logic in an assembler application. The application will read each LREC in the subfile until it

finds a record that contains any of the keys in the key list. See “Using Boolean Logic in Key Lists” on page 30 for more information about how to use Boolean operators in key lists.

```
SW01SR REG=R5
LA    R5,EBX000                    LOAD BASE OF DBKEY TABLE
XC    EBX000(L'SW01NKY+2*L'SW01KIT),EBX000  CLEAR KEYLIST AREA
MVC   SW01NKY,=H'2'
DBSETK BASE=R5,KEYNUM=1,DIS=I/2,LEN=I/1,CON=#DF_EQ,MSK=X'80', *
      ID1=#DF_UP+#DF_CONST,ID2=#DF_OR
DBSETK BASE=R5,KEYNUM=2,DIS=I/6,LEN=I/4,SEA=EBW000,ID1=#DF_UP
DBKEY  REF=IR73DF,KEYLIST=EBX000    ACTIVATE KEY LIST
DBRED  REF=IR73DF
```

- The following example reads an extended LREC that has:
  - A primary key of X'80'
  - A specific 4-character department number in the subLREC field PR17DNR
  - A specific 10-character employee name in the userLREC field PR17NAM.

```
MVC EBW000(4),=C'TZ35'           Department number
MVC EBW004(10),=C'HARRISON  '    Name of employee

DBRED ...,KEY1=(PKY=#PR17K80),KEY2=(T=PR17DNR,S=EBW000),KEY3=(R=PR17NAM,S=EBW004)
```

- The following example reads a variable-length LREC that has:
  - A primary key of X'80'
  - A specific 7-character flight number in field PR25FLT
  - A specific 6-character board and off point in field PR25BRD.

```
MVC EBW000(L'PR20FLT),PR20FLT         Flight number
MVC EBW007(L'PR20BRD),PR20BRD         Board and off point

DBRED ...,KEY1=(PKY=#PR25K80),KEY2=(R=PR25FLT,S=EBW000,L=L'PR20FLT+L'PR20BRD)
```

Related Macros

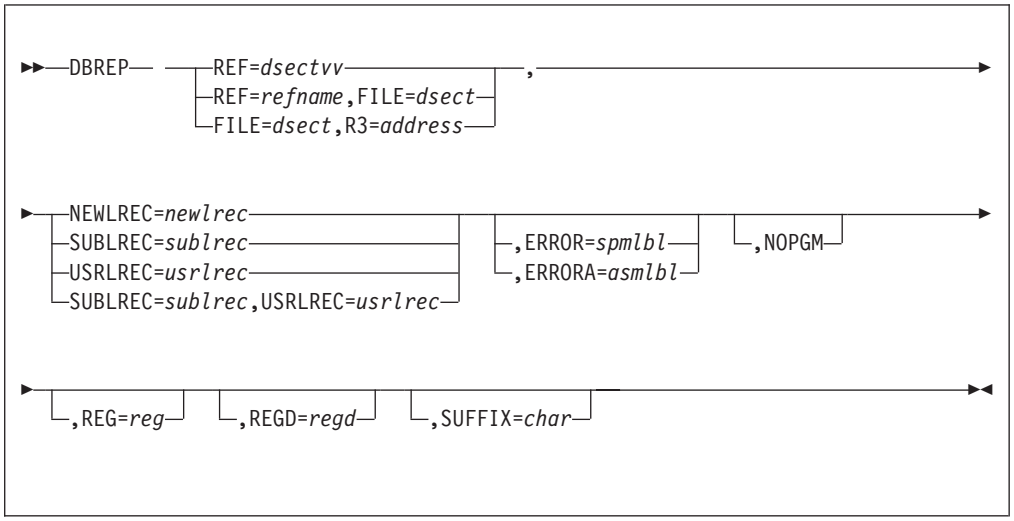
- “DBADD–Add a Logical Record to a Subfile” on page 176
- “DBKEY–Activate a Key List” on page 247.

DBREP–Replace a Logical Record with Another Logical Record

Use this macro to replace the following:

- A previously read logical record (LREC) with a new LREC
- The userLREC in the current extended LREC with a new userLREC
- The subLREC in the current extended LREC with a new subLREC.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

A*I**refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

NEWLREC=*newlrec*
replaces a new fixed-length or variable-length LREC, where *newlrec* is one of the following:

- A register that contains the address of the LREC

- A label in one of the following formats:

  newlrec
  is the label of a field that contains the LREC.

  AI*newlrec*
  is the label of a 4-byte field that contains the storage location of the LREC.

SUBLREC=*sublrec*
replaces a subLREC in the current extended LREC with another subLREC, where *sublrec* is one of the following:

- A register that contains the address of the subLREC
- A label in one of the following formats:

  sublrec
  is the label of a field that contains the subLREC.

  AI*sublrec*
  is the label of a 4-byte field that contains the storage location of the subLREC.

USRLREC=*usrlrec*
replaces the userLREC in the current extended LREC with another userLREC, where *usrlrec* is one of the following:

- A register that contains the address of the userLREC
- A label in one of the following formats:

  usrlrec
  is the label of a field that contains the userLREC.

  AI*usrlrec*
  is the label of a 4-byte field that contains the storage location of the userLREC.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

NOPGM
specifies not to change the program stamp in a block when filing it.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

Before specifying the SUBLREC parameter, you must specify a DBRED macro to locate the current LREC.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- When using the DBREP macro with the SUBLREC parameter, the current extended LREC must contain one (and only one) subLREC. When an extended LREC contains more than one subLREC, use DBDEL to delete one or more subLRECs; then call DBADD to add each new subLREC.
- The new LREC can be larger or smaller or the same size as the old LREC (if you have defined variable-length LRECs in the DSECT).
- If the subfile is UP or DOWN organized, do not use DBREP to insert an LREC containing key definitions that are different from those in the LREC that it is replacing. This would destroy the subfile organization. Instead, use DBDEL to delete the original LREC and DBADD to add the new LREC. DBADD allows you to specify the keys in the LREC that you add to the subfile.
- If the subfile uses block index support, the TPFDF product automatically updates the block index when you replace an LREC.
- For P-type files, the data is copied into the current block.
- The DBREP macro replaces the current LREC. That is, the DBREP macro does not perform an internal DBRED macro.
- Do not use the DBREP macro if you have changed:
  - Any key fields
  - Any fields in the LREC that are also used as index key fields.

  Instead, delete the old LREC with a DBDEL macro and add a new LREC with a DBADD macro.

Examples

- The following example replaces the current LREC in file GR23DF with the LREC GR23REC.

```
DBREP REF=GR23DF,NEWLREC=GR23REC
```

- The following example replaces a subLREC in the current extended LREC with the subLREC at the address contained in register 5.

```
DBREP REF=GR39DF,SUBLREC=R5
```

- The following example replaces a subLREC at the same time as a userLREC.

```
DBREP REF=GR39DF,USRLREC=GR39REC,SUBLREC=A/EBW000
```

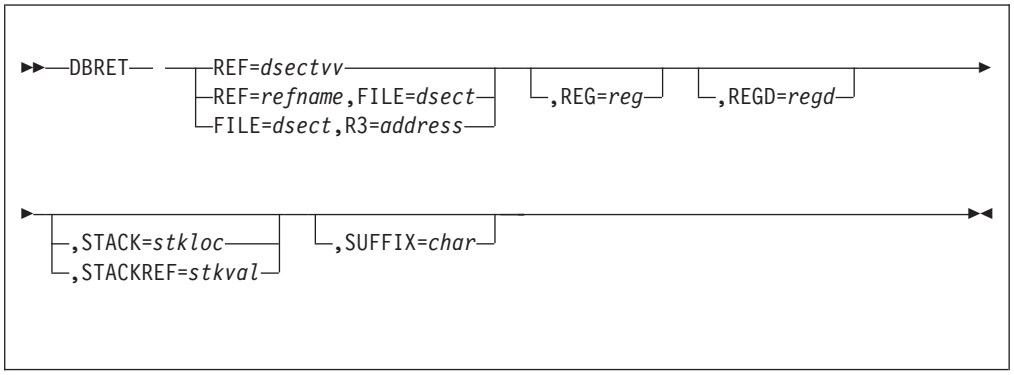
Related Macros

- “DBADD–Add a Logical Record to a Subfile” on page 176
- “DBDEL–Delete One or More Logical Records” on page 214
- “DBMOD–Perform or Indicate Logical Record Modifications” on page 249
- “DBRED–Read a Logical Record” on page 272.

DBRET–Retain a Logical Record Position

Use this macro to retain the file address and displacement in a block of the current logical record (LREC). You can read this LREC later in your application program by using a DBRED macro with a PREVIOUS, STACK, or STACKREF parameter.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

STACK=*stkloc*
retains an LREC, where *stkloc* is the location of a 10-byte field that contains the details about the LREC. Specify one of the following:

- A register that points to the location of the field that contains the LREC details.
- A label in one of the following formats:

  stkloc
  is the label of the field that contains the LREC details.

  AI*stkloc*
  is the label of a 4-byte field that contains the storage address of that contains the LREC details.

The 10-byte field contains the following information:
- 4 bytes for the current file address
- 4 bytes for the current core address
- 2 bytes for the next available byte (NAB) value.

Note: Where possible, use the STACKREF parameter.

STACKREF=*stkval*
retains an LREC, where *stkval* is a value assigned to the LREC. Specify one of the following:

- A 4-byte value
- A register that contains the 4-byte value
- An absolute value that does not exceed 4-bytes.

The retained information consists of the following 14-bytes:
- 4 bytes for the current file address
- 4 bytes for the current core address
- 2 bytes for the next available byte (NAB) value
- 4 bytes for the STACKREF value.

Note: Where possible, use the STACKREF parameter instead of the STACK parameter.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

Before using this macro, you must open the subfile using the DETAC or HOLD parameter of the DBOPN macro.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can

code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.

- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- If you do not specify the STACK or STACKREF parameter, the details of only the current LREC are retained.
- To ensure that an LREC is retrieved accurately when you use the DBRED macro, do not use the DBRET macro with the STACK and STACKREF parameters specified on the same open file. If you do, the wrong LREC could be retrieved. You must close and reopen the file each time you alternate between specifying the STACK and STACKREF parameters.

Examples

- The following example retains the file address of file IWA1DF.

```
DBRET REF=IWA1DF,REG=R5
```

- The following example retains the address of the current LREC in file GR25DF and later reads that LREC.

```
DBRET REF=GR25DF,STACKREF==F'5'
   :
DBRED REF=GR25DF,STACKREF==F'5'
```

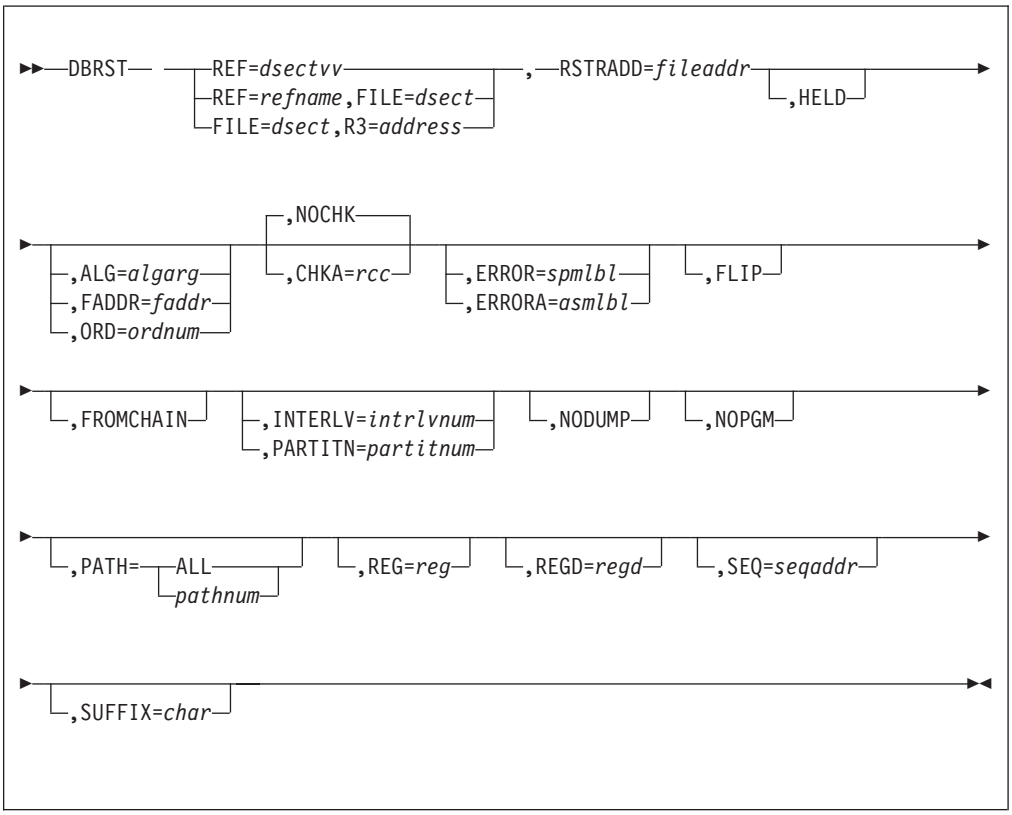
Related Macros

“DBRED–Read a Logical Record” on page 272.

DBRST–Restore a Subfile

Use this macro to restore a subfile (previously copied by the DBCPY macro) to a file address that you specify.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

RSTRADD=*fileaddr*
specifies where you want to restore the current subfile, where *fileaddr* is the label of a 4-byte field containing the address.

If this address is set to X'00000000', the DBRST macro creates a new subfile and restores the current subfile in the new subfile. The address of the prime block of the new subfile is placed in the SW00FAD field of the SW00SR.

HELD
restores a subfile that is held; that is, one that was opened using the DBOPN macro with the HOLD parameter specified.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FLIP
exchanges the contents of 2 subfiles rather than overwrites the original subfile with its modified copy.

If you specify the FLIP parameter, the DBRST macro transfers the subfile currently located at the file address you specify with the RSTRADD parameter with the subfile that is currently open.

The SW00SR now references the subfile specified with RSTRADD rather than the subfile that was originally opened.

FROMCHAIN
restores only the prime block of the subfile. Any overflow blocks in the subfile are chained to this new prime block without copying the overflow blocks to new pool blocks.

INTERLV=*intrlvnum*
specifies the number of the interleave that you want to use, where *interlvnum* is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN=*partitnum*
specifies the number of the partition that you want to use, where *partitnum* is one of the following:

- A register that contains the address of the partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm calculates the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

NODUMP
specifies that you do not want the TPFDF product to issue any of the following system errors while processing this macro:

- DB0100
- DB0102
- DB0117
- DB0123
- DB0138
- DB0140.

See *Messages (System Error, Online, Offline) and Master Glossary* for more information about these system errors.

Note: Using the NODUMP parameter is not recommended because it can prevent system errors from being issued that indicate a critical problem.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

SEQ=*seqaddr*
specifies an update sequence number, where *seqaddr* is the address of a 2-byte field that contains the number. The number you provide must match the sequence number contained in the subfile that is specified by the RSTRADD parameter.

If the numbers do not match, the DBRST macro does not proceed and issues an error return. If the numbers match, the DBRST macro restores the subfile and increases the sequence number by 1. This sequence number is placed in the prime block of the restored subfile.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

Before you can use the DBRST macro, you must make a copy of a subfile using the DBCPY macro. You can modify this copy using other TPFDF macros.

Normal Return

The restored file (with any modifications you have made to the copy) becomes the currently open subfile. You can continue processing it using other TPFDF macros. The copy is released unless you use the FLIP parameter.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.
- When you use the SEQ parameter, the DBRST macro returns an error if the number specified and the sequence number in the file specified by RSTRADD do not match. Bit 6 is set in the SW00RTN field and the subfile is not restored.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- The DBRST macro rebuilds the B$^+$Tree index for B$^+$Tree files.

Examples

- The following example exchanges the contents of subfile GR37DF with the contents of the subfile referenced by EBW030. The SW00SR now references the subfile at EBW030 rather than the subfile that was originally opened.

```
DBRST REF=GR37DF,RSTRADD=EBW030,FLIP
```

- The following example restores a subfile using a sequence number.

```
DBRST REF=GR19DF,RSTRADD=EBW030,SEQ=EBW020
```

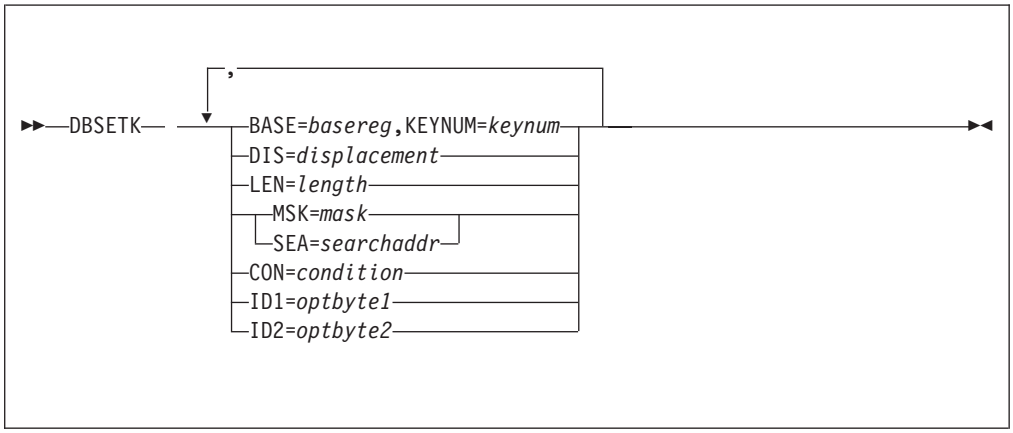
Related Macros

“DBCPY–Copy a Subfile” on page 205.

DBSETK–Setting Up a Key in a Key List

Use this macro to set up a key list. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

Format



Note: Parameters can be entered as labels, registers, or immediate values as described in the following section. (Immediate values are assumed to be decimal unless otherwise specified.) For example:

Label	DIS=EBW000
Register	DIS=R5
Immediate value	DIS=I/16

BASE=*basereg*
specifies the base address of the key list, where *basereg* is a register that contains the address. If you do not specify this parameter, it is assumed that the register with addressability to SW01SR is set up so that the SW01SR DSECT overlays the key to be updated.

KEYNUM=*keynum*
specifies the number of the key that you are setting up, where *keynum* is the label of an area, a register, or an immediate value that contains a number from 1–180, depending on the type of key list. See “Specifying Logical Records (LRECs) Using Keys” on page 19 for more information about key lists.

DIS=*displacement*
specifies the displacement of the key field in the LREC from the start of the LREC, where *displacement* is the label of an area, a register, or an immediate value that contains the displacement in bytes. For example, if the LREC is a variable-length LREC and you want to specify the LREC ID as the key field, the value is 2. The maximum displacement you can specify with this macro is 255.

LEN=*length*
specifies the length of the key field in the LREC, where *length* is the label of an area, a register, or an immediate value that contains the length in bytes. For example, if you want to specify the LREC ID as the key field, the value is 1. If you are using a mask field, *length* must be 1.

MSK=*mask*
specifies a mask value or 1-byte search argument, where *mask* is the label of an area, a register, or an immediate value that contains a 1-byte mask or a 1-byte search argument.

SEA=*searchaddr*
specifies a search argument, where *searchaddr* is the label of an area that contains the search argument or a register that contains the address of the search argument.

CON=*condition*
specifies the condition that must exist for the match to be successful, where *condition* is the label of an area, a register, or an immediate value that contains the value. Use one of the following values if you specify the #DF_CONST, #DF_CHAR, or #DF_PACKED value for the ID1 parameter:

#DF_GT
specifies that the LREC key field is greater than the search argument. The corresponding hexadecimal value is X'20'.

#DF_GE
specifies that the LREC key field is greater than or equal to the search argument. The corresponding hexadecimal value is X'40'.

#DF_EQ
specifies that the LREC key field is equal to the search argument. The corresponding hexadecimal value is X'70'.

#DF_NE
specifies that the LREC key field is not equal to the search argument. The corresponding hexadecimal value is X'80'.

#DF_LE
specifies that the LREC key field is less than or equal to the search argument. The corresponding hexadecimal value is X'B0'.

#DF_LT
specifies that the LREC key field is less than the search argument. The corresponding hexadecimal value is X'D0'.

Use one of the following values if you specify the #DF_MASK value for the ID1 parameter:

#DF_NM
specifies that the result is not mixed ones and zeros (that is, all zeros or all ones are found). The corresponding hexadecimal value is X'10'.

#DF_NO
specifies that the result is not all ones. The corresponding hexadecimal value is X'40'.

#DF_Z
specifies that the result is all zeros. The corresponding hexadecimal value is X'70'.

#DF_NZ
specifies that the result is not all zeros. The corresponding hexadecimal value is X'80'.

#DF_M
specifies that the result is mixed ones and zeros. The corresponding hexadecimal value is X'B0'.

#DF_O
specifies that the result is all ones. The corresponding hexadecimal value is X'E0'.

ID1=*optbyte1*
specifies the settings for the first option byte, where *optbyte1* is the label of an area, a register, or an immediate value that contains the indicators for the first option byte. Use one or more of the following values:

#DF_DOWN
specifies that the subfile is DOWN organized on this key field (bit 0 = 1).

#DF_UP
specifies that the subfile is UP organized on this key field (bit 1 = 1).

#DF_KEYS
specifies that you want to use default keys (bit 2 = 1).

#DF_USR
specifies the userLREC portion of an extended LREC (bit 3 = 1).

#DF_CONST
specifies that the MSK parameter contains a 1-byte search argument (bit 4 = 1).

#DF_PACKED
specifies that the search argument is a variable-length packed string (bit 5 = 1).

#DF_MASK
specifies that the MSK parameter contains a 1-byte mask (bit 6 = 1).

#DF_NOORG
specifies that the subfile is not organized on this key field. This value is equated to X'00'.

#DF_CHAR
specifies that the search argument is a variable-length character string. This value is equated to X'00'.

ID2=*optbyte2*
specifies either a Boolean connector or a global modification operation, where *optbyte2* is the label of an area, a register, or an immediate value that contains the indicators for the second option byte.

Use one of the following values for a Boolean connector:

#DF_OR
specifies the OR connector (bit 0 = 1).

#DF_AND
specifies the AND connector (bit 1 = 1).

#DF_ORIF
specifies the ORIF connector (bit 4 = 1).

#DF_ANDIF
specifies the ANDIF connector (bit 5 = 1).

Use one of the following values for a global modification operation:

#DF_MVI
moves the value contained in SW01MSK into the LREC at the displacement specified by SW01DIS. The corresponding hexadecimal value is X'04'.

#DF_MVC
moves the character string whose address is in SW01SEA into the LREC, starting at the displacement specified by SW01DIS for the length contained in SW01LEN. The corresponding hexadecimal value is X'08'.

#DF_FILL
propagates the character contained in SW01MSK into the LREC, starting at the displacement specified by SW01DIS for the length contained in SW01LEN. The corresponding hexadecimal value is X'0C'.

#DF_OI
performs an OR-Immediate (OI) operation on the byte in the LREC at the displacement in SW01DIS by using the value specified in SW01MSK. The corresponding hexadecimal value is X'10'.

#DF_OC
performs an OR-Character (OC) operation in the LREC beginning on the byte whose displacement is specified in SW01DIS for a length given in SW01LEN by using the value whose address is given in SW01SEA. The corresponding hexadecimal value is X'14'.

#DF_NI
performs an AND-Immediate (NI) operation on the byte in the LREC at the displacement in SW01DIS by using the value specified in SW01MSK. The corresponding hexadecimal value is X'18'.

#DF_NC
performs an AND-Character (NC) operation in the LREC beginning on the byte whose displacement is specified in SW01DIS for a length given in SW01LEN by using the value whose address is given in SW01SEA. The corresponding hexadecimal value is X'1C'.

#DF_XI
performs an Exclusive OR-Immediate (XI) operation on the byte in the LREC at the displacement in SW01DIS by using the value specified in SW01MSK. The corresponding hexadecimal value is X'20'.

#DF_XC
performs an Exclusive OR-Character (XC) operation in the LREC beginning on the byte whose displacement is specified in SW01DIS for a length given in SW01LEN by using the value whose address is given in SW01SEA. The corresponding hexadecimal value is X'24'.

#DF_ADD
adds the fullword value whose address is in SW01SEA to the fullword value in the LREC whose displacement is specified by SW01DIS. The corresponding hexadecimal value is X'28'.

#DF_AH
adds the halfword value whose address is in SW01SEA to the halfword value in the LREC whose displacement is specified by SW01DIS. The corresponding hexadecimal value is X'2C'.

#DF_SUB
subtracts the fullword value whose address is in SW01SEA to the fullword value in the LREC whose displacement is specified by SW01DIS. The corresponding hexadecimal value is X'30'.

#DF_SH
subtracts the halfword value whose address is in SW01SEA to the halfword

value in the LREC whose displacement is specified by SW01DIS. The corresponding hexadecimal value is X'34'.

#DF_CNT
increments a fullword counter whose address is in SW01SEA. The application is responsible for initializing the counter before the global modification operation. The corresponding hexadecimal value is X'38'.

#DF_CTH
increments a halfword counter whose address is in SW01SEA. The application is responsible for initializing the counter before the global modification operation. The corresponding hexadecimal value is X'3C'.

#DF_SUM
adds the fullword value in the LREC at the displacement specified in SW01DIS to the fullword sum whose address is in SW01SEA. The sum must be initialized before the global modification operation. The corresponding hexadecimal value is X'40'.

#DF_SMH
adds the halfword value in the LREC at the displacement specified in SW01DIS to the halfword sum whose address is in SW01SEA. The sum must be initialized before the global modification operation. The corresponding hexadecimal value is X'44'.

#DF_MAX
finds the maximum value of the fullword value in the LREC at the displacement specified by SW01DIS and the current fullword maximum whose address is in SW01SEA. The new maximum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation. The corresponding hexadecimal value is X'48'.

#DF_MXH
finds the maximum value of the halfword value in the LREC at the displacement specified by SW01DIS and the current halfword maximum whose address is in SW01SEA. The new maximum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation. The corresponding hexadecimal value is X'4C'.

#DF_MIN
finds the minimum value of the fullword value in the LREC at the displacement specified by SW01DIS and the current fullword minimum whose address is in SW01SEA. The new minimum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation. The corresponding hexadecimal value is X'50'.

#DF_MNH
finds the minimum value of the halfword value in the LREC at the displacement specified by SW01DIS and the current halfword minimum whose address is in SW01SEA. The new minimum value is stored at the address in SW01SEA. The value in SW01SEA does not have to be initialized before the global modification operation. The corresponding hexadecimal value is X'54'.

Entry Requirements

The SW01SR DSECT must be addressable by a register without a suffix. If you specify the BASE parameter, the SW01SR DSECT must be addressable without a suffix by the register passed to the BASE parameter.

Normal Return

None.

Error Return

None.

Programming Considerations

- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- Registers R14 and R15 cannot be used when a register is specified as a parameter.
- This macro does not set field SW01NKY. This has to be done on a separate instruction.
- A key list field in the SW01SR DSECT is modified or filled in only if its corresponding parameter is supplied.
- If you specify a label for the KEYNUM, DIS, or LEN parameters, TPFDF assumes the area is a 2-byte location. If you specify a label for the CON, MSK, ID1, or ID2 parameters, TPFDF assumes the area is a 1-byte location.

Examples

See “Using a Key List with the DBSETK Macro” on page 28 for an example of using the DBSETK macro.

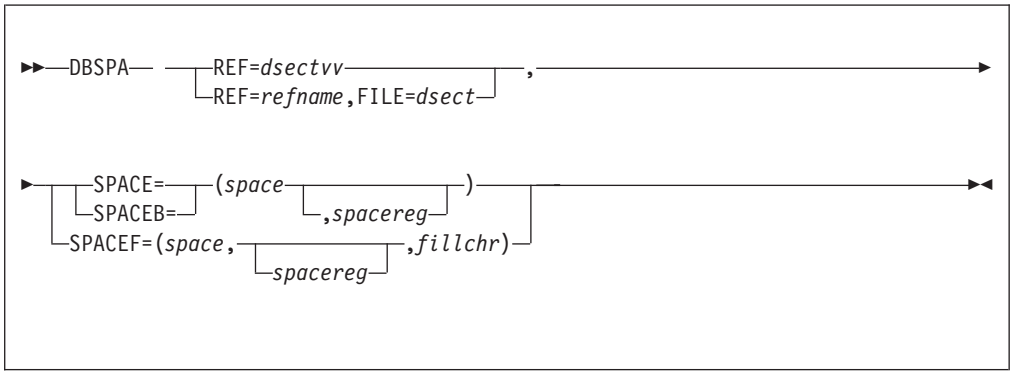
Related Macros

“DBKEY–Activate a Key List” on page 247.

DBSPA–Create Work Space

Use this function to obtain and initialize work space linked to the SW00SR slot for a subfile. This space is available while the subfile is open.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

SPACE
allocates work space for an open subfile and initializes the work space to X'00'

SPACEB
allocates work space for an open subfile and initializes the work space to X'40'

SPACEF
allocates work space for an open subfile and initializes the work space to a specified character.

space
is the number of bytes of space that you want, which can be a maximum of 3952 bytes. Specify one of the following:

- A register that contains the number of bytes of space that you want.
- An absolute value.
- The length of a label.
- The label of a field. The space is determined by the 2 bytes of data beginning at the specified label.

If you request a zero amount of space, the DBSPA macro releases any space that was previously allocated for this subfile.

spacereg
is the register in which you want the base address of the work space loaded.

fillchr
is the character you want to use to initialize the work space. Specify one of the following:

- The label of a 1-byte field that contains the character.
- An immediate value (for example, =X'FF').

Entry Requirements

None.

Normal Return

The address of the space that the TPFDF product has provided. The TPFDF product also loads this address into field SW00WKA in the SW00SR slot.

Error Return

None.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- If you specified a SPACE or SPACEB parameter with the DBOPN macro and you then call the DBSPA macro with the same REF parameter value, the TPFDF product returns the space originally allocated.

Examples

- The following example creates a 400-byte area filled with space characters (X'40').

```
DBSPA REF=GR36DF,SPACEB=(400,R5)
```

- The following example provides space that is the same length as field GR36REC.

```
DBSPA REF=GR36DF,SPACE=(L'GR36REC,R5)
```

- The following example provides an amount of space determined by the 2 bytes of data beginning at label EBW002.

```
DBSPA REF=GR36DF,SPACE=(EBW002,R5)
```

- The following example releases any space that was previously allocated for subfile GR33DF.

```
DBSPA REF=GR33DF,SPACE=(0)
```

- The following example creates a 400-byte area filled with character X'FF'.

```
DBSPA REF=GR36DF,SPACEF=(400,,=X'FF')
```

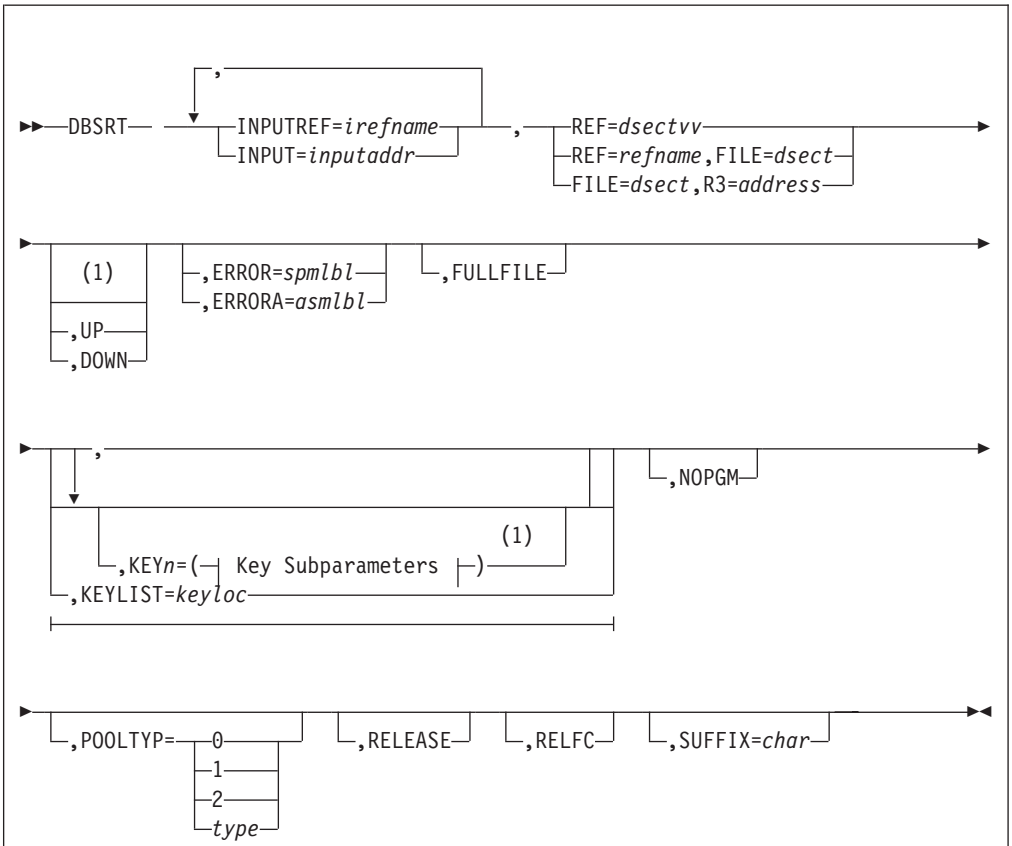
Related Macros

"DBOPN–Open a Subfile" on page 260.

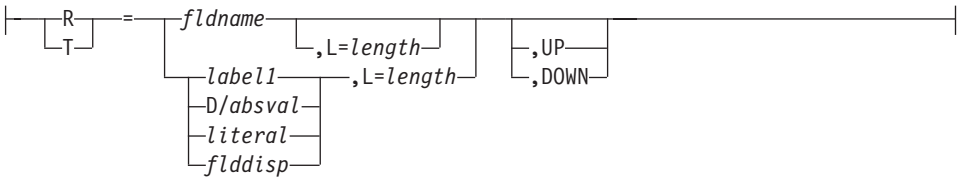
DBSRT–Sort a Subfile

Use this macro to sort logical records (LRECs) in an open subfile.

Format



Key Subparameters:



Notes:

1 See “Specifying File Organization with Key*n* Parameters” on page 23 for information about the rules for using the KEY*n* parameters and file organization parameters together.

INPUTREF=*irefname*

specifies the reference name of the input subfile, where *irefname* is one of the following:

- The DSECT name

- A label that references the DSECT name in one of the following formats:

  irefname
  is the label of an 8-byte field containing the DSECT name.

  A*irefname*
  is the label of a 4-byte field that contains the storage address of an 8-byte field containing the DSECT name.

INPUT=*inputaddr*
specifies the base address of the SW00SR slot of the input file.

Note: This parameter is supported for migration purposes only; where possible, always use the INPUTREF parameter to identify the input subfile.

REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

A*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

UP
specifies that the LRECs are organized in the subfile in ascending order of key fields.

DOWN
specifies that LRECs in the subfile are organized in descending order of key fields.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FULLFILE
sorts LRECs from the entire input file (not from a single subfile) to the output subfile specified with the REF parameter.

KEY*n*
specifies the key parameters that you want to use with this macro, where *n* is a number from 1–6. You can specify as many as six KEY*n* parameters and they must be specified in sequential order beginning with 1. That is, you cannot code a KEY2 parameter without a KEY1 parameter, a KEY3 parameter without the KEY1 and KEY2 parameters, and so on.

If you use these parameters, you must also specify the file organization of the keys. See “Specifying File Organization with Key*n* Parameters” on page 23 for more information about how to do this. Use one or more of the following subparameters with the KEY*n* parameter:

R specifies a field in the LREC to be used for placing the LRECs in the output file.

T specifies a field in the subLREC of an extended LREC to be used for placing the LRECs in the output file.

fldname
is the name of a field defined in the DSECT for the LREC; for example:

```
... KEY1=(R=GR00FLD)
```

label1
is a 2-byte field containing the displacement into the LREC; for example:

```
... KEY1=(R=EBX010,L==H'4')
```

D*I**absval*
specifies the displacement into the LREC of the field, where *absval* is an absolute value; for example:

```
... KEY1=(R=D/2,L=L'GR00NAM,UP)
```

You can also specify the absolute value implicitly; for example:

```
... KEY1=(R=D/GR00NAM-GR00REC,L=L'GR00NAM,UP)
```

literal
is a halfword literal containing the displacement into the LREC; for example:

```
... KEY1=(R==H'2',L==H'4')
```

flddisp
is the displacement off the field of the LREC; for example:

```
... KEY1=(R=GR00FLD+2,L==H'4')
```

or

```
... KEY1=(R=GR00FLD+L'GR00FLD,L==H'4')
```

L=*length*
specifies the length of the field to be used in placing the LRECs in the output file, where *length* is one of the following:

- The label of a 2-byte field containing the length of the field
- A 2-byte literal
- An absolute value in the form of L'*fldname* (for example, L=L'GR92FLD).

The default value is the length of the field specified with the R subparameter.

UP
specifies that the key field is in ascending order in the subfile.

DOWN
specifies that the key field is in descending order in the subfile.

KEYLIST=*keyloc*
specifies a key list that you want to use with this macro, where *keyloc* is one of the following:

- A register that contains the address of the key list
- A label in one of the following formats:

  keyloc
  is a label indicating the address of the key list.

  AI*keyloc*
  is the label of a 4-byte field that contains the storage address of the key list.

See “Setting Up and Using a Key List” on page 26 for information about how to set up a key list.

NOPGM
specifies not to change the program stamp in a block when filing it.

POOLTYP
overrides the pool type defined by the database administrator, where:

0 uses the pool type defined by the PF0 parameter of the DBDEF macro.

1 uses the pool type defined by the PF1 parameter of the DBDEF macro.

2 uses the pool type defined by the PF2 parameter of the DBDEF macro.

type
is the label of a 1-byte field that contains a 0, 1, or 2 to specify to pool type.

Use the POOLTYP parameter as directed by the database administrator.

RELEASE
releases the SW00SR slot for the input subfile after the macro has completed processing.

RELFC
releases the input subfile and deletes it from DASD. All overflow blocks are released. If the file is a pool file, the prime block is also released. If the file is a fixed file, the prime block is initialized to empty.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

Both subfiles must be opened before you call the DBSRT macro.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent macros when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with macros that access or update records without using fullfile processing.
- Any keys that are active when you call this macro are used to select records from the input file.
- If the output subfile is not a B+Tree subfile, you ***must*** specify keys using KEY*n* parameters or the KEYLIST parameter when you call this macro. The keys specify how the TPFDF product sorts the LRECs in the output subfile.

  The KEY*n* and KEYLIST parameters are ignored for B+Tree files. The output file is organized according default keys defined on the DBDEF macro for the file. See *TPFDF Database Administration* for more information about default keys.
- The TPFDF product creates the output subfile for you when using a W- or R-type input file. You can use standard DSECT SR05SR, which is provided with the TPFDF product, for the output file if the input subfile contains variable-length LRECs.

  If you do not use SR05SR, you must specify a subfile of the same type as the input file.
- The DBSRT macro does not change the input subfile. The TPFDF product clears the ECB level for the input subfile once the DBSRT macro has been processed.
- After processing, the original contents of the output subfile are lost.
- When this macro has completed processing, the output subfile is left open and must be closed using the DBCLS macro before the ECB exits. If you specify the RELEASE parameter, this macro closes the input subfile.
- If you sort a large file in detac mode, the TPFDF product puts the sorted output in a newly created pool file and closes the output file that was originally specified. You must restore the file to a fixed file to permanently save the results.
- You cannot issue additional TPFDF macros to the input file until the file is closed if:
  - You specify the FULLFILE parameter,
  - You do not specify the RELEASE parameter, and
  - The end-of-file indicator is set.

However, you can specify the REUSE parameter on the DBCLS macro. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.

- You cannot use this macro with P-type files, add current files, or pushdown chaining files.
- Figure 21 shows how the DBSRT macro sorts LRECs from one subfile into another.

Figure 21. Sorting LRECs from One Subfile into Another. The input file is defined by the INPUTREF parameter and the output file is defined by the REF parameter.

- If you use the DBSRT macro in a commit scope, both the input and output files must be opened in the same commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.
- When you use the DBSRT macro to sort large input files, use detac mode for output files to ensure optimum system performance.

Examples

- The following example sorts an input file called IWTLDC into SR05SR and changes the reference name.

```
DBOPN REF=SR05SR,HOLD,DETAC
DBSRT REF=SR05SR,ERROR=label,RELFC,                          *
      INPUTREF=IWTLDC,RELEASE,                               *
      KEY1=(R=D/IWTLKEY-IWTLREC,UP),                         *
      KEY2=(R=D/IWTLAPI-IWTLREC,L==AL2(IWTLLDT-IWTLAPI),UP)
DBIFB REF=SR05SR,NEWREF=IWTLDC
```

- The following example uses an output file other than SR05SR.

```
DBOPN REF=IR00DFX,HOLD,DETAC,POOLTYP=1
DBSRT REF=IR00DFX,RELFC,ERROR=label,                         *
      INPUTREF=IR00DF,RELEASE,                               *
      KEY1=(PKY=#IR00K80,UP),                                *
      KEY2=(R=IR00FLD,L=EBW000,UP)
```

- The following is another example using an output file other than SR05SR.

```
DBOPN REF==C'SR05SR  ',FILE=IWTLDC
DBSRT REF==C'SR05SR  ',INPUTREF=IWTLDC,                      *
      KEY1=(PKY=#IWTKPKY,UP),                                *
      KEY2=(R=IWTLAPI,L==AL2(IWTLLDT-IWTLAPI),UP)
```

Related Macros

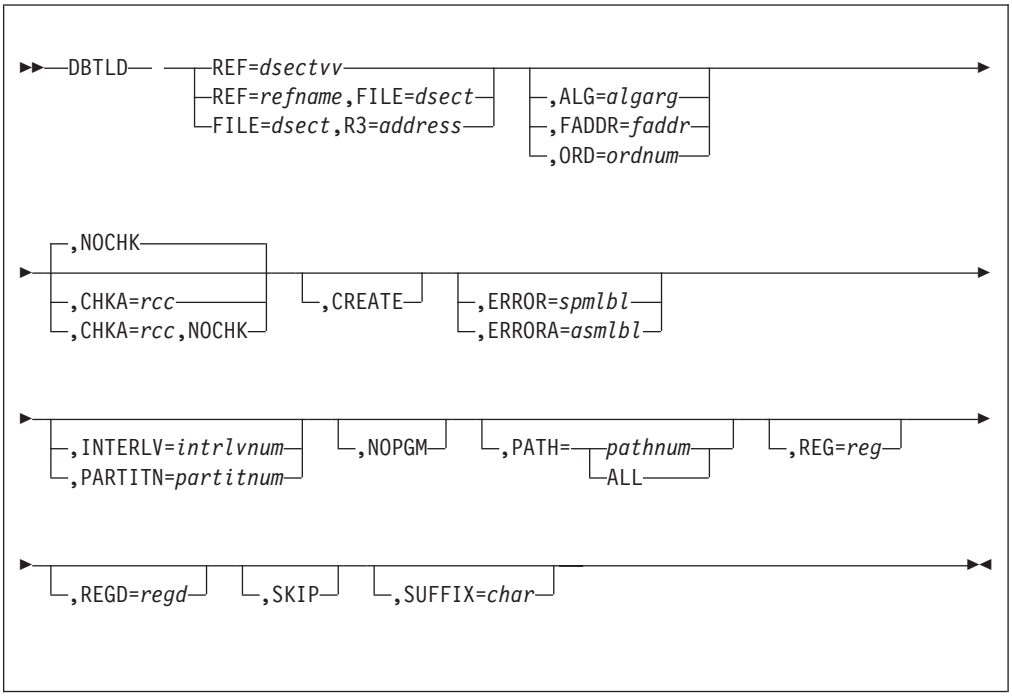
“DBMRG–Merge Logical Records from Two Subfiles” on page 254.

DBTLD–Write a Subfile from Main Storage to DASD

Use this macro to do one of the following:

- Write the subfile to DASD
- Ignore the subfile.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

CREATE
creates a new subfile when writing the file blocks to DASD instead of writing the blocks back to their original file addresses.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

INTERLV=*intrlvnum*
specifies the number of the interleave that you want to use, where *interlvnum* is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN=*partitnum*
specifies the number of the partition that you want to use, where *partitnum* is one of the following:
- A register that contains the address of the partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm calculates the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

NOPGM
specifies not to change the program stamp in a block when filing it.

PATH
specifies the path for a detail subfile using index support. If there is only one index path, do not specify this parameter. Specify one of the following:

pathnum
is the path number or the label of a 2-byte field that contains the path number. The number of index paths used is defined by your database administrator.

ALL
specifies all paths.

See *TPFDF Database Administration* for more information about path numbers.

REG=*register*
specifies a register in which to return the address of the current LREC (this address is contained in SW00SR field SW00REC). You must specify this parameter for T-type files.

REGD=*register*
specifies a register in which to return the base address of the userLREC part of an extended LREC.

SKIP
discards the blocks that were read from tape or sequential data set with the DBTRD macro. The DBTLD macro releases all the blocks, both prime and chained, that the DBTRD macro retrieved and placed in main storage.

You can use the SKIP parameter in a restart situation when a number of blocks need to be read from tape or sequential data set to reach the point where a system failure occurred. (All the blocks up to the failure point have already been written to DASD, so you only need to read them without saving them again.)

The SKIP parameter is also useful if you want to end the transfer of information from tape to disk, or if there are unwanted blocks of data on a tape or sequential data set.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

You must successfully read a subfile using the DBTRD macro before calling the DBTLD macro.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.

- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- The following rules determine the value of the record code check (RCC) value used when the TPFDF product creates a new subfile:
  - If you do not specify the NOCHK parameter, the TPFDF product creates new subfiles with a random RCC value.
  - If you specify the NOCHK parameter without the CHKA parameter, the TPFDF product creates new subfiles without an RCC value.
  - If you specify both the NOCHK and CHKA parameters, the TPFDF product creates new subfiles with the RCC value specified with the CHKA parameter.
- You must always call DBTLD after calling DBTRD. No other macro calls are allowed between the DBTRD and DBTLD calls. Make sure your application checks for any error conditions before invoking DBTLD. During DBTRD processing, if an error occurs that causes SW00RTN to be nonzero, invoking DBTLD is treated as an invalid command sequence.
- Because the DBTLD macro requires a significant amount of system resources, do not use this macro in a commit scope. See “Commit Scopes” on page 8 for more information about commit scopes.
- The DBTLD macro rebuilds the B^+Tree index for B^+Tree files.

Examples

The following example writes a subfile to new pool blocks in DASD.

```
DBTLD REF=GR24DF,CREATE
```

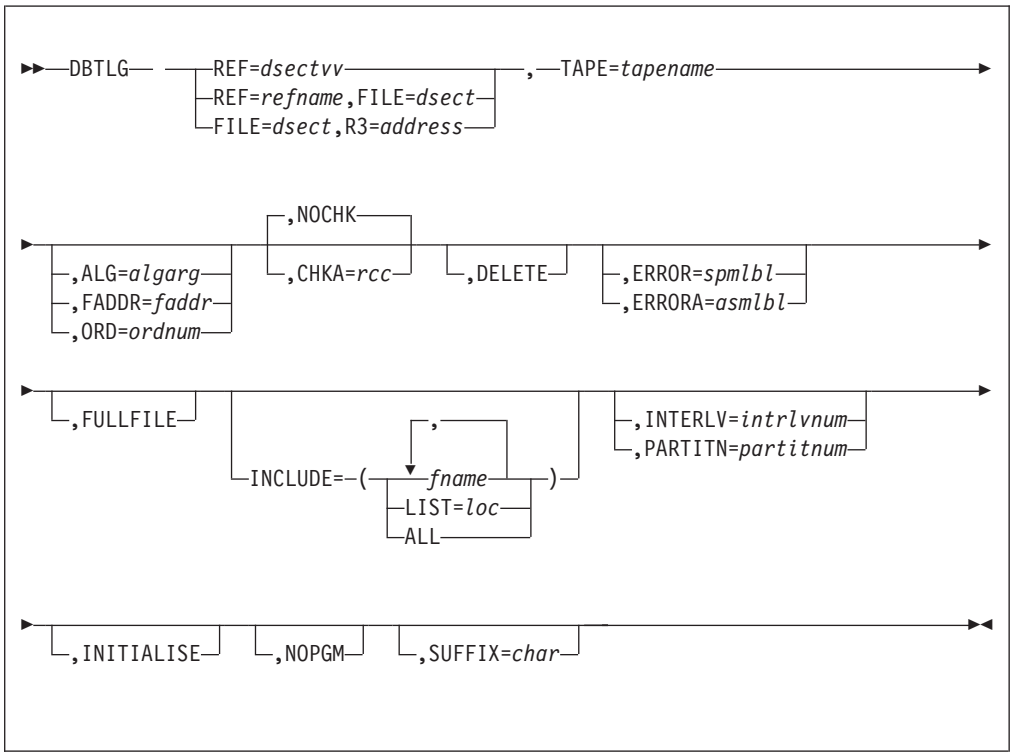
Related Macros

- “DBTRD–Read a Subfile from an Input Tape to Main Storage” on page 323
- “DBTLG–Write a File or Subfile to Tape” on page 318.

DBTLG–Write a File or Subfile to Tape

Use this macro to write a file, or part of a file, to a real-time tape, a general tape, or a sequential data set.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

TAPE=*tapename*
specifies the tape or sequential data set to which you want the file to be written, where *tapename* is a 3-character symbolic tape name.

ALG=*algarg*
identifies the subfile that you want to access, where *algarg* specifies an algorithm argument.

The TPFDF product uses the algorithm argument to determine the subfile (ordinal number) that is to be accessed. Specify the algorithm argument based on the type of algorithm that is defined in the DSECT or DBDEF macro for the file. If the DSECT or DBDEF macro defines the #TPFDB04 or the #TPFDB0D algorithm, do not use this parameter.

If the subfile you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the TPFDF product uses the algorithm argument to locate the subfile. See *TPFDF Database Administration* for more information about how the TPFDF product uses the algorithm argument to locate the subfile.

Specify *algarg* as one of the following:

- A register that contains the address of the algorithm argument
- A literal value that specifies the algorithm argument (for example, ALG==C"SMITH")
- A label in one of the following formats:

  algarg
  is the label of a field that contains the algorithm argument.

  AI*algarg*
  is the label of a 4-byte field that contains the storage address of the algorithm argument.

Note: Do not modify the area of storage containing the algorithm argument until the subfile is closed.

FADDR=*faddr*
identifies the subfile that you want to access, where *faddr* is one of the following:

faddr
is the label of a 4-byte field that contains the file address of the prime block of the subfile.

AI*faddr*
is the label of a 4-byte field that contains the storage address of the file address of the prime block of the subfile.

ORD=*ordnum*
identifies the subfile that you want to access, where *ordnum* is one of the following:

ordnum
is the label of a 4-byte field that contains the ordinal number of the subfile.

AI*ordnum*
is the label of a 4-byte field that contains the storage address of the ordinal number of the subfile.

If the file is partitioned or interleaved, specify the relative ordinal number within the partition or interleave. If the file is not partitioned or interleaved, specify the file address compute program (FACE) ordinal number.

CHKA=*rcc*
checks the record code check (RCC) value in each block, where *rcc* is the label of a 1-byte field that contains the RCC character.

NOCHK
specifies that you do not want to check the record code check (RCC) value of the blocks.

DELETE
deletes all the LRECs in the file after the file is written to tape or sequential data set. Any previously chained blocks are returned to pool. If the file is a pool file, all prime blocks are released.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

FULLFILE
writes LRECs from the whole input file (not from a single subfile) to tape or sequential data set.

INCLUDE
writes additional subfiles to tape or sequential data set. The DBTLG macro writes the subfiles to tape or sequential data set in sequence.

fname
is the name of the additional subfile or subfiles that you want to write to tape.

LIST=*loc*
specifies a list of subfiles that you want to write to tape in addition to the file or subfile specified with the REF parameter, where *loc* is the location of a 22-byte field containing a list of as many as 10 file IDs. Specify one of the following:

loc
is the label of the field that contains the list.

A*loc*
is the label of a 4-byte field containing the storage address of the field that contains the list.

The list starts with a 2-byte field that contains a halfword count of the number of file IDs referenced, each subsequent 2-byte field contains a file ID.

Note: The list contains the file IDs, not the file names.

ALL
writes all the detail subfiles referenced by LRECs in the main subfile or file to tape.

INTERLV=*intrlvnum*
specifies the number of the interleave that you want to use, where *interlvnum* is one of the following:
- A register that contains the address of the interleave number
- An absolute value representing the interleave number
- The label of a 2-byte field that contains the interleave number.

If you specify this parameter, the maximum interleave number must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about interleaves.

PARTITN=*partitnum*
specifies the number of the partition that you want to use, where *partitnum* is one of the following:
- A register that contains the address of the partition number
- An absolute value representing the partition number
- The label of a 2-byte field that contains the partition number.

If you specify this parameter, the number of partitions and the end ordinal must be defined in the DSECT or DBDEF macro. See *TPFDF Database Administration* for more information about partitions.

Note: Do not use this parameter with the #TPFDB0F algorithm. This algorithm calculates the partition used from the algorithm argument. See *TPFDF Database Administration* for more information about algorithms.

INITIALISE
initializes the DASD file after writing it to tape. Any blocks that are chained to the prime block are released.

All the LRECs in the prime block are deleted, and the next available byte (NAB) in the header indicates the first byte directly after the header.

The TPFDF product retains the prime block of a pool file. The prime block consists of a block that is empty, apart from the header and file identifier.

NOPGM
specifies not to change the program stamp in a block when filing it.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can

code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.

- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- After opening a file, if you use fullfile processing to access or update records, you must continue to use fullfile processing on any subsequent macros when available until the file is closed. Accessing or updating subfiles using fullfile processing cannot be mixed with macros that access or update records without using fullfile processing.
- If you use the FULLFILE parameter, and the end-of-file indicator is set, you cannot issue additional TPFDF macros until the file is closed. However, you can specify the REUSE parameter on the DBCLS macro. See “Identifying Return Indicators and Errors” on page 13 for information about the end-of-file indicator.
- If you specify the INCLUDE and DELETE parameters on the same DBTLG macro statement, all the subfiles referenced by the INCLUDE parameter are deleted after they have been written to tape or sequential data set. If you specify INCLUDE=(ALL) and the DELETE parameter, all the subfiles belonging to a particular file (as specified by the REF parameter) are deleted after they are written to tape.
- The DBTLG macro copies the blocks to tape or sequential data set in sequence.
- The DBTLG macro does not write the B^+Tree index information to tape because it is rebuilt when the B^+Tree data file is restored.
- If you use the DBTLG macro in a commit scope, a rollback of the commit scope will not restore the contents or the position of the tape. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

- The following example writes file GR54DF to tape VPH.

```
DBTLG REF=GR54DF,TAPE=VPH
```

- The following example writes the GR20DF subfile to tape along with the two additional subfiles specified with the INCLUDE parameter.

```
DBTLG REF=GR20DF,INCLUDE=(GR21DF,GR22DF)
```

- The following example writes the GR22DF subfile to tape along with detail subfiles reference from GR22DF that have file ID X'FE30' or X'FE31'.

```
MVC   EBX000(2),=H'2'                 Count of file IDs in list
MVC   EBX002(2),=X'FE30'              Add file ID to list
MVC   EBX004(2),=X'FE31'              Add file ID to list
DBTLG REF=GR22DF,INCLUDE=(LIST=EBX000)
```

- The following example writes all the detail subfiles referenced by LRECs in file GR21DF.

```
DBTLG REF=GR21DF,INCLUDE=(ALL)
```

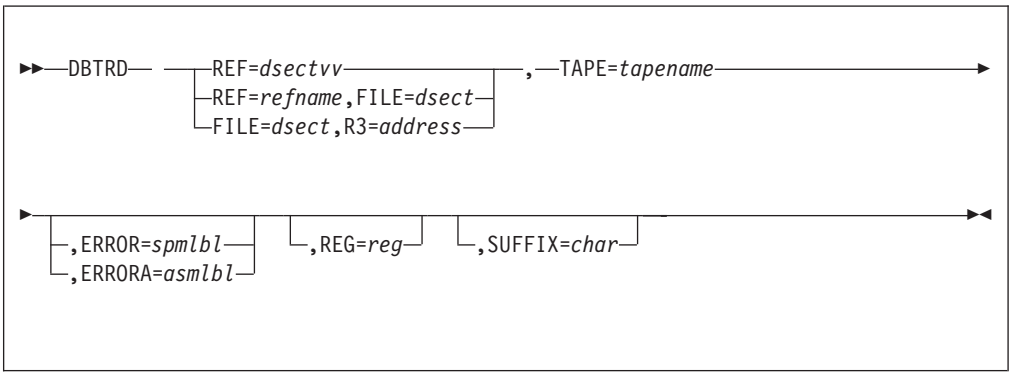
Related Macros

- “DBTLD–Write a Subfile from Main Storage to DASD” on page 313
- “DBTRD–Read a Subfile from an Input Tape to Main Storage” on page 323.

DBTRD–Read a Subfile from an Input Tape to Main Storage

Use this macro to read a subfile from an input tape or sequential data set to main storage.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

R3=*address*
specifies the location of the SW00SR slot for this subfile, where *address* is the label of a field that contains the address of the SW00SR slot. Register 3 will be loaded with this address.

Note: Do not use this parameter; it is provided only for migration purposes. Use the REF parameter to specify the file that you want to access.

TAPE=*tapename*
specifies the tape from which you want to read the data, where *tapename* is a 3-character symbolic tape name or sequential data set.

ERROR=*spmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *spmlbl* is a TPFDF structured program macro (SPM) label defined with the #LOCA macro. See *TPFDF and TPF Structured Programming Macros* for more information about the #LOCA macro. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

ERRORA=*asmlbl*
branches to the specified location if a serious error is detected when processing the macro, where *asmlbl* is an assembler label. See “Identifying Return Indicators and Errors” on page 13 for more information about serious errors.

REG=*register*
specifies a register in which to return the address of the prime block (this address is contained in SW00SR field SW00PCA).

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

None.

Normal Return

None.

Error Return

See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.
- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- The DBTRD macro reads one subfile at a time. After you call the DBTRD macro, you must then call the DBTLD macro before reading another subfile. (You can use the SKIP parameter with the DBTLD macro to ignore the subfile.)
- If you use the DBTRD macro in a commit scope, a rollback of the commit scope will not restore the position of the tape. See “Commit Scopes” on page 8 for more information about commit scopes.

Examples

The following example reads subfile GR54DF from tape TST to main storage.

```
DBTRD REF=GR54DF,TAPE=TST
```

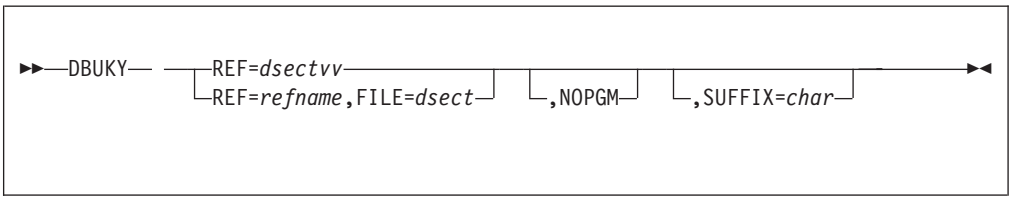
Related Macros

- “DBTLD–Write a Subfile from Main Storage to DASD” on page 313
- “DBTLG–Write a File or Subfile to Tape” on page 318.

DBUKY–Generate a Unique Key for Use in Logical Records

Use this macro to generate a unique key in the SW00SR slot for the subfile.

Format



REF=*dsectvv*
specifies the file or subfile that you want to access, where *dsectvv* is the DSECT name and an optional 2-character version.

REF=*refname*
specifies the file or subfile that you want to access, where *refname* is a label that references the DSECT name in one of the following formats:

refname
is the label of an 8-byte field that contains the 6-byte DSECT name and an optional 2-character version.

AI*refname*
is the label of a 4-byte field that contains the storage address of the DSECT name and an optional 2-character version.

FILE=*dsect*
specifies the file or subfile that you want to access, where *dsect* is the DSECT name.

NOPGM
specifies not to change the program stamp in a block when filing it.

SUFFIX=*char*
allows you to use the same DSECT to map two different areas of storage, where *char* is the suffix character.

Entry Requirements

None.

Normal Return

The new unique key value is returned in the 4-byte SW00UKY field of the SW00SR.

Error Return

- See “Identifying Return Indicators and Errors” on page 13 for information about how to check the error indicators.
- A zero value in the SW00UKY field of the SW00SR indicates an error.

Programming Considerations

- The optional 2-character version on the REF parameter allows you to access more than one subfile in the same file at the same time. For example, you can code REF=IR71DF01,ALG==C"A" to access subfile A and REF=IR71DF02,ALG==C"B" to access subfile B.

- If you specify a label, the label must be more than 3 characters long.
- The contents of register 14 (R14) and R15 cannot be predicted across a TPFDF macro call.
- The contents of R3, which contains the storage address of the SW00SR slot, are used by TPFDF macro calls. Do not change the value of R3 between macro calls unless you save the value after each macro call and restore the value before each macro call.
- The value that is returned in the SW00UKY field of the SW00SR is only valid immediately after the DBUKY macro is processed. Subsequent TPFDF macros reuse and overwrite this field.
- In order to use the unique key feature, the file header must be expanded by 18 bytes.
- See “Grouping LRECs Together Using the Unique Key Facility” on page 6 for more information about using unique keys.

Examples

The following example generates a unique key value for file GR34DF. The value is placed in the SW00UKY field in the SW00SR slot.

```
DBUKY REF=GR34DF
```

The following is an example of a header expanded by 18 bytes so that the unique key feature can be used.

```
IRXXHDR& DS  CL16         STANDARD FILE HEADER
         DS  CL10         STANDARD TPFDF HEADER
         DS  CL18         UNIQUE KEY HEADER EXTENSION
```

Related Macros

“DBKEY–Activate a Key List” on page 247.

TPFDF Restricted Assembler Macros: Reference

This chapter documents Product-Sensitive Programming Interface information.

The macros in this section (with the exception of FMSGS) are TPFDF internal macros used to perform system functions. There is no guarantee that the programming interface to these macros will not change. The use of these macros should be restricted to minimize the effect of any changes.

The FMSGS macro is provided for compatibility with older applications. Use the TPF or ALCS WTOPC macro instead of the FMSGS macro whenever possible.

This chapter contains an alphabetic listing of the TPFDF restricted assembler macros. The description of each macro includes the following information:

Format: Provides a syntax (railroad track) diagram for the macro and a description of each parameter and variable. See “How to Read the Syntax Diagrams” on page xiv for more information about syntax diagrams.

Entry Requirements: Lists any special conditions that must be true when you use the macro.

Normal Return: Lists what is returned when the macro has completed processing successfully.

Error Return: Lists what is returned when the macro cannot complete processing successfully.

Programming Considerations: Lists any additional considerations for using the macro, including any restrictions or limitations.

Examples: Provides one or more examples that show you how to code the macro.

Related Macros: Lists where to find information about related macros.

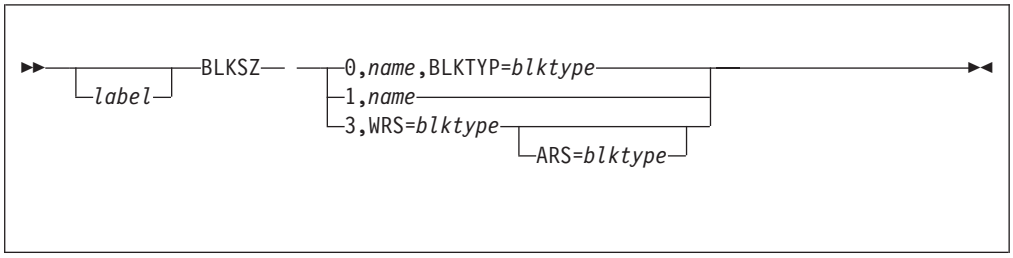
© Copyright IBM Corp. 1997, 2001

BLKSZ–Convert a Block Type to a Block Size

Use this macro to set global variables at assembly time based on specified block types. You can set the following information:

- Block size
- Maximum next available byte (NAB)
- Control bits.

Format



label
is a symbolic name assigned to the macro statement.

0 specifies the processing mode that is used to get the physical block size and control bits corresponding to the specified block type.

1 specifies the processing mode that is used to get the physical block size, maximum next available byte (NAB), and control bits corresponding to block types specified by global variables &SW00WRS and &SW00ARS.

3 specifies the processing mode that is used to get the physical block size, maximum NAB, and control bits corresponding to the block types specified by the WRS and ARS parameters.

name
is the name of the segment or macro that called the BLKSZ macro. If an error occurs while processing the BLKSZ macro, the specified name is displayed in the assembly error message.

BLKTYP
specifies the block type for which the physical block size and control bits will be set.

WRS
specifies the prime block type for which the physical block size, maximum NAB, and control bits will be set.

ARS
specifies the overflow block type for which the physical block size, maximum NAB, and control bits will be set. If you do not specify this parameter or if you set it to 0, the values set will be the same as those set for the WRS parameter.

blktype
is one of the following block types:

L0 specifies a 128-byte block size. Do not use this value when you specify processing mode 3.

L1 specifies a 381-byte block size.

L2 specifies a 1055-byte block size.

L3 specifies a 4000-byte block size. This block type is available only in an ALCS environment.

L4 specifies a 4095-byte block size.

L5 specifies a user-defined size. This block type is available only in an ALCS environment.

L6 specifies a user-defined size. This block type is available only in an ALCS environment.

L7 specifies a user-defined size. This block type is available only in an ALCS environment.

L8 specifies a user-defined size. This block type is available only in an ALCS environment.

Entry Requirements

If you specify processing mode 1, you must set the assembler global variable &SW00WRS (and optionally, variable &SW00ARS) to a valid block type.

Normal Return

- If you specify processing mode 0, assembler global variables are set as follows for the specified block type:

Variable	Set To
&BLKSZS	The physical block size.
&BLKSZPI	The control bits.

- If you specify processing mode 1, assembler global variables are set as follows for the block type specified by assembler global variable &SW00WRS:

Variable	Set To
&BLKSZPS	The physical block size.
&BLKSZPN	The maximum NAB.
&BLKSZPI	The control bits.

  In addition, assembler global variables are set as follows for the block type specified by assembler global variable &SW00ARS:

Variable	Set To
&BLKSZAS	The physical block size.
&BLKSZAN	The maximum NAB.
&BLKSZAI	The control bits.

  Note: If variable &SW00ARS is 0 or not defined, these variables are set to the same values that were used for &SW00WRS.

- If you specify processing mode 3, assembler global variables are set as follows for the block type specified by the WRS parameter:

Variable	Set To
&BLKSZPS	The physical block size.
&BLKSZPN	The maximum NAB.
&BLKSZPI	The control bits.

In addition, assembler global variables are set as follows for the block type specified by the ARS parameter:

Variable	Set To
&BLKSZAS	The physical block size.
&BLKSZAN	The maximum NAB.
&BLKSZAI	The control bits.

Note: If you do not specify the ARS parameter, these variables are set to the same values that were used for the WRS parameter.

Error Return

If you specify an incorrect processing mode or block type, an assembler error (referred to as an MNOTE) with a severity of 8 is issued.

Programming Considerations

- This macro does not generate any object code. All calculations and error checking are performed at assembly time.
- The value of the control bits are set as follows for each block type:
  - In a TPF system:

Block Type	Control Bit Setting
L0	X'00'
L1	X'00'
L2	X'10'
L4	X'20'

  - In an ALCS environment:

Block Type	Control Bit Setting
L0	X'00'
L1	X'00'
L2	X'10'
L3	X'20'
L4	X'30'
L5	X'40'
L6	X'50'
L7	X'60'
L8	X'70'

  The control bits are used in field STDCTL in the standard header of blocks. See the STDHD DSECT or C$STDHD header file for more information.
- Register values are preserved across this macro call.

Examples

- The following example returns the physical block size and control bits for an L1 block type.

```
BLKSZ 0,GRT8SR,BLKTYP=L1
```

The following assembler global variables will be set by the macro:

Variable	Setting
&BLKSZS	381
&BLKSZPI	X'00'

- The following example returns the physical block size, maximum NAB, and control bits for L2 and L4 block types.

```
&SW00WRS SETC 'L2'
&SW00ARS SETC 'L4'
   ⋮
BLKSZ 1,IR75DF
```

The following assembler global variables will be set by the macro:

Variable	Setting
&BLKSZPS	1055
&BLKSZPN	1019
&BLKSZPI	X'10'
&BLKSZAS	4095
&BLKSZAN	4059
&BLKSZAI	X'20' (in a TPF system) or X'30' (in an ALCS environment)

- The following example returns the physical block size, maximum NAB, and control bits for the L2 block type.

```
BLKSZ 3,IR80DF,WRS=L2
```

The following assembler global variables will be set by the macro:

Variable	Setting
&BLKSZPS	1055
&BLKSZPN	1019
&BLKSZPI	X'10'
&BLKSZAS	1055
&BLKSZAN	1019
&BLKSZAI	X'10'

Related Macros

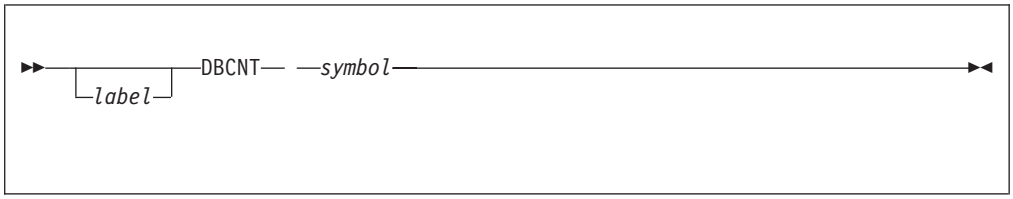
None.

DBCNT–Calculate the Length of an Assembler Symbol

This macro calculates the length of an assembler symbol.

Note: Do not use this macro in new code; use assembler count attribute K' in new code.

Format



label
is a symbolic name assigned to the macro statement.

symbol
is an assembler symbol.

Entry Requirements

You must define symbol &KK to the assembler by coding:

```
GBLA  &KK
```

Normal Return

Symbol &KK is set to the length of the specified symbol.

Error Return

None.

Programming Considerations

Register values are preserved across this macro call.

Examples

The following example sets symbol &KK to the length of symbol &INCLUDE.

```
DBCNT &INCLUDE
```

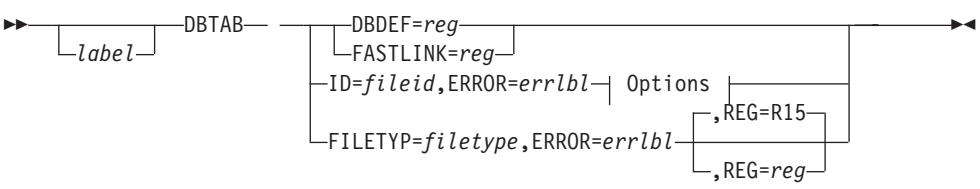
Related Macros

None.

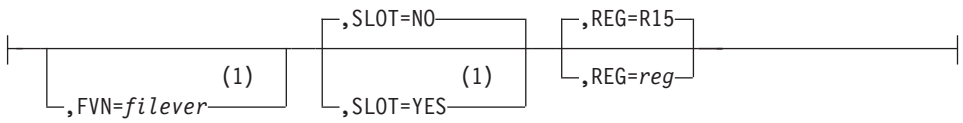
DBTAB–Access Database Definition (DBDEF) Tables

Use this macro to access the database definition (DBDEF) tables.

Format



Options:



Notes:

1 If you specify SLOT=YES, you cannot specify the FVN parameter.

label
is a symbolic name assigned to the macro statement.

DBDEF=*reg*
returns the address of the base of the DBDEF index table to a specified register, *reg*.

FASTLINK=*reg*
returns the address of the base of the fast-link table to a specified register, *reg*.

ID=*fileid*
specifies the file identifier (ID) for a specific file, where *fileid* is the label of a 2-byte field containing the file ID.

ERROR=*errlbl*
specifies that label to which you want to branch if:

- The specified file ID is not defined in the DBDEF tables.
- The specified file ID is not in the range of valid file IDs. The range is specified by variable #TPFDBID in the ACPDBE macro. See *TPFDF Installation and Customization* for more information about the ACPDBE macro.

FVN=*filever*
specifies the file version number of the file ID specified by the ID parameter, where *filever* is the label of a 1-byte field containing the file version number.

SLOT
specifies one of the following:

YES
returns the address of the slot that contains the address of the DBDEF table for a specific file ID.

NO
returns the address of the DBDEF table for a specific file ID.

REG=*reg*
specifies a register, *reg*, in which the requested address is returned.

FILETYP=*filetype*
returns the address of a DBDEF table for a specific record type, where *filetype* is the label of a 2-byte field containing the record type. The record type is represented with a numeric value defined in the TPF SYSEQC macro or ALCS DXCURID macro.

Entry Requirements

None.

Normal Return

The requested address is returned in the specified register.

Error Return

None.

Programming Considerations

- The DBTAB macro modifies register 14 (R14). In addition, if you specify the FVN parameter, this macro modifies R7.
- Figure 22 shows the different areas that you can access with the DBTAB macro.

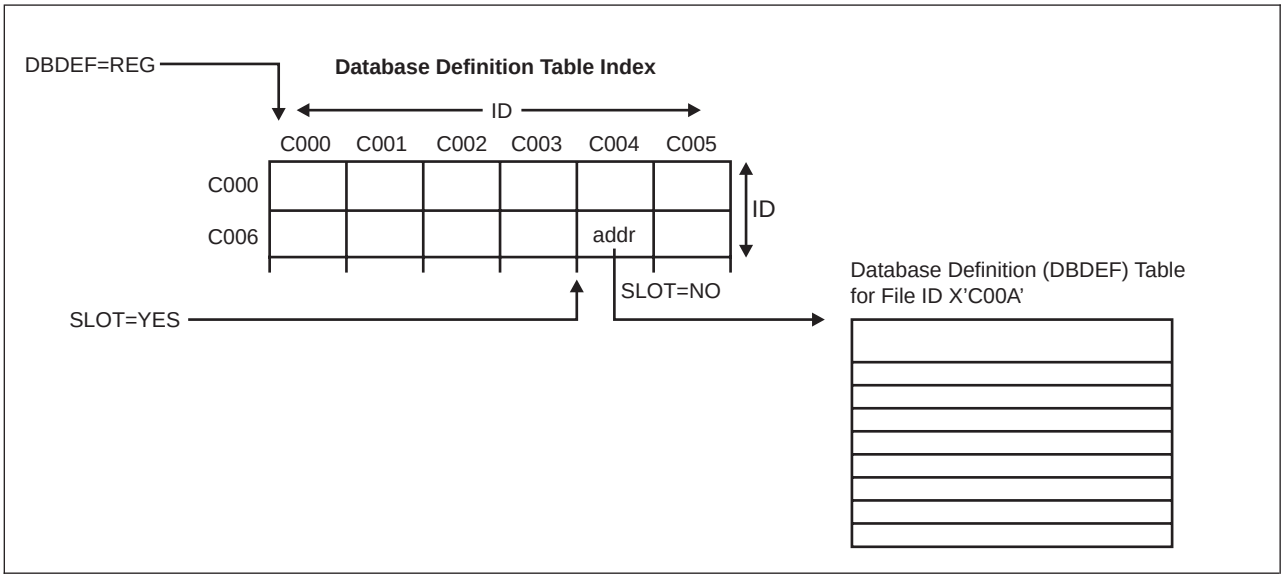


Figure 22. Areas in the Database Definition (DBDEF) Table Accessed by the DBTAB Macro

Examples

- The following example returns the base of the DBDEF index table to R15.

```
DBTAB DBDEF=R15
```

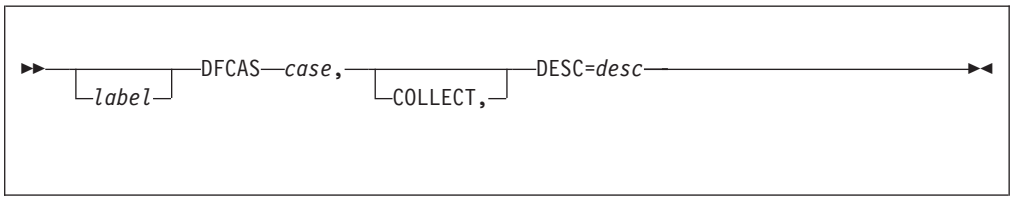
- The following example returns the base of a specific file ID to R15. In this example, the file ID is located in BFAID and the file version of the file is located in EBX000.

```
DBTAB ID=BFAID,FVN=EBX000,ERROR=NODBDEF
```

DFCAS–TPFDF Case Setup in Fast-link Segments

Use this macro to set up a fast-link case in a fast-link segment.

Format



label
is a symbolic name assigned to the macro statement.

case
is the case number, which is a decimal number in the range 0–7. The case number is in the segment specified using the DFLNK macro with the *prog* parameter.

COLLECT
specifies that the fast-link case is included in the TPFDF data collection.

Note: A data collection case number equate has to be provided when you specify the COLLECT parameter; for example:

```
#UWAACASE0 EQU 0
```

DESC=*desc*
specifies the description of the function for a case number, where *desc* is the description of the function. You can display this description by using the ZUDFM OAI/DBTAB command with the LINK parameter specified. See *TPFDF Utilities* for more information about the ZUDFM OAI/DBTAB command.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

- You must specify the DFCAS command at the beginning of every fast-link segment.
- When you specify the DFCAS command, you must provide an assembler equate to mark the start of the case; for example:

  ```
  #CASE0 EQU *
  ```

Examples

The following example defines case numbers 0 to 3; case number 1 and case number 2 are included in the TPFDF data collection:

```
DFCAS 0,DESC='SEARCH DATA BLOCK'
DFCAS 1,COLLECT,DESC='FILNC DD'
DFCAS 2,COLLECT,DESC='FILEC DD'
DFCAS 3,DESC='UPDATE TRAILER'
```

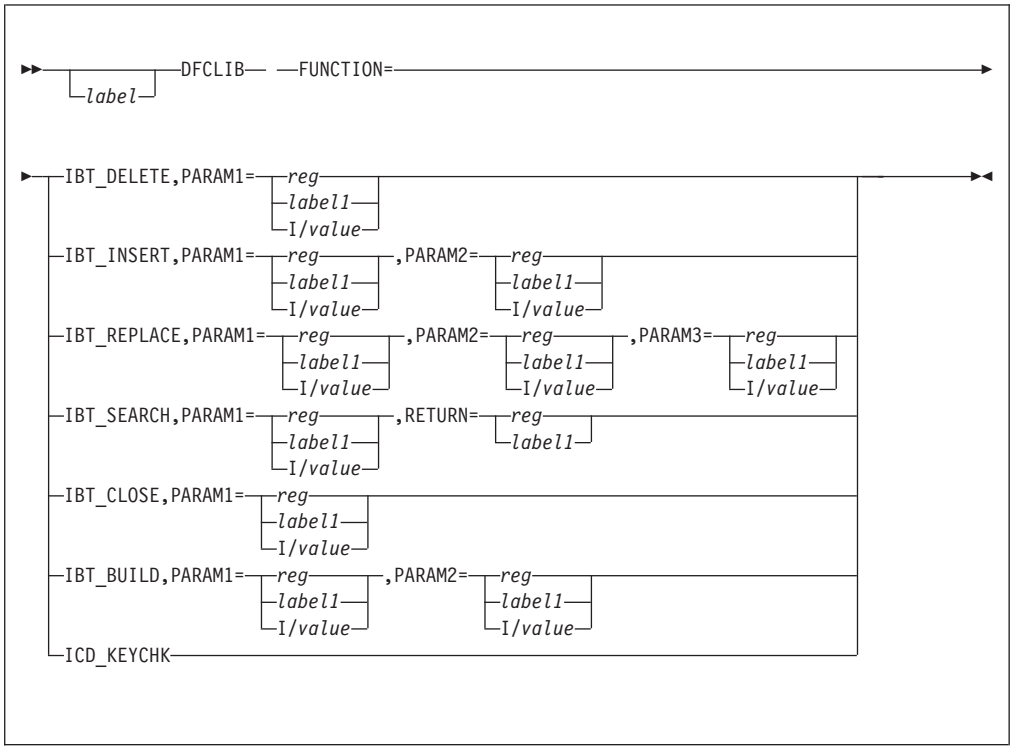
Related Macros

None.

DFCLIB–C Language Interface

This macro provides an interface from TPFDF assembler programs to TPFDF C language programs. Use this macro in TPFDF assembler programs to make a single line call to TPFDF C language code. This macro simplifies coding and provides one central point that can be updated if the interface changes.

Format



label
is a symbolic name assigned to the macro statement.

reg
is a register that contains the appropriate value based on the specified parameter.

label1
is a label that contains the appropriate value based on the specified parameter.

I/*value*
specifies an immediate value based on the specified parameter.

IBT_DELETE
deletes a key from the B+Tree index when the last logical record (LREC) in the block is deleted, where:

PARAM1
specifies the core address of the first LREC in the block that is being deleted.

IBT_INSERT
inserts a key into the B+Tree index when a data block is split or when a new data block is added, where:

PARAM1
specifies the core address of the first LREC in the new data block.

PARAM2
specifies the file address of the new data block.

The default keys contained in the first LREC of the new data block are inserted into the B+Tree index.

IBT_REPLACE
replaces a key in the B+Tree index when the first LREC in a data block is deleted and there are more LRECs in the same data block, or when a new LREC becomes the first LREC in the data block, where:

PARAM1
specifies the core address of the *old* first LREC in the data block; that is, the LREC that is to be replaced.

PARAM2
specifies the file address of the data block.

PARAM3
specifies the core address of the *new* first LREC in the data block; that is, the LREC that is to replace the LREC specified by the PARAM1 parameter.

IBT_SEARCH
locates a data block using the B+Tree index, where:

PARAM1
specifies the core address of the LREC that is being added or deleted during an add or delete operation. If this is not an add or delete operation, the parameter is ignored.

RETURN
specifies the location of the data block file address or zero.

IBT_CLOSE
specifies one of the following:
- Commit or stop DETAC mode updates to B+Tree index nodes.
- Close an entire B+Tree index for a subfile following a DBCLS macro.

PARAM1
specifies one of the following values:

IBT_CLOSE_FINAL
commits DETAC mode updates.

IBT_CLOSE_ABORT
stops DETAC mode updates and prevents any updates made in DETAC mode from being written to DASD.

IBT_CLOSE_SUBFILE
closes a B+Tree index.

IBT_BUILD
inserts one data block into a B+Tree index when rebuilding the entire B+Tree index, where:

PARAM1
specifies the core address of the first LREC in the data block.

PARAM2
specifies the file address of the data block.

ICD_KEYCHK
Verifies that global modification does not attempt to change the default keys of a logical record (LREC).

Entry Requirements

SW00SR REG=R3 must be declared before the macro call with register 3 (R3) set to be the base of the SW00SR DSECT.

Normal Return

SW00RET bit 0 and SW00RT2 bit 4 return zero.

Error Return

SW00RET bit 0 and SW00RT2 bit 4 return an error indicator.

Programming Considerations

- The contents of R0–R7 are preserved across this macro call.
- The contents of R14 and R15 cannot be predicted.
- You must code a DFCLIB macro with the IBT_BUILD parameter for every data block in a subfile when rebuilding a B+Tree index. In addition, the data blocks must be passed to the DFCLIB macro in a specific order. Overflow blocks must be passed first, in order, according to their forward chain fields. The prime block must then be passed as the last data block to the DFCLIB macro.

Examples

- The following example deletes a key from the B+Tree index. R5 contains the core address of the first LREC in the block being deleted.

  ```
  DFCLIB FUNCTION=IBT_DELETE,PARAM1=R5
  ```

- The following example inserts a key in the B+Tree index. R5 contains the core address of the first LREC in the new data block. Field EBW000 contains the file address of the new data block.

  ```
  DFCLIB FUNCTION=IBT_INSERT,PARAM1=R5,PARAM2=EBW000
  ```

- The following example commits any DETAC mode updates to the B+Tree index nodes.

  ```
  DFCLIB FUNCTION=IBT_CLOSE,PARAM1=I/IBT_CLOSE_FINAL
  ```

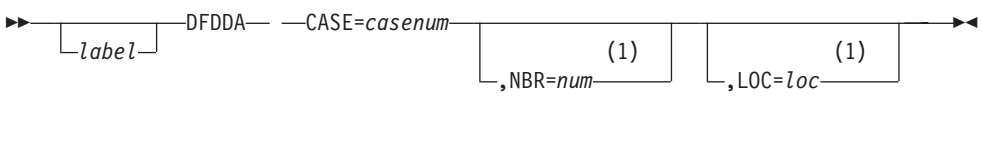
Related Macros

None.

DFDDA–Distributed Data Access Support

Use this macro to implement the TPFDF Distributed Data Access (TPFDF/DDA) feature.

Format



Notes:

1 If you specify CASE=2, you must code this parameter. If you do not specify CASE=2, this parameter is ignored.

label
is a symbolic name assigned to the macro statement.

CASE=*casenum*
specifies the case number passed with the macro call, where *casenum* is a decimal number.

NBR=*num*
specifies the intercept number, where *num* is a decimal number from 0–4.

LOC=*loc*
specifies the intercept location, where *loc* is a decimal number from 0–4.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

Register values are preserved across this macro call.

Examples

The following example calls DFDDA case 3.

```
DFDDA CASE=3
```

Related Macros

None.

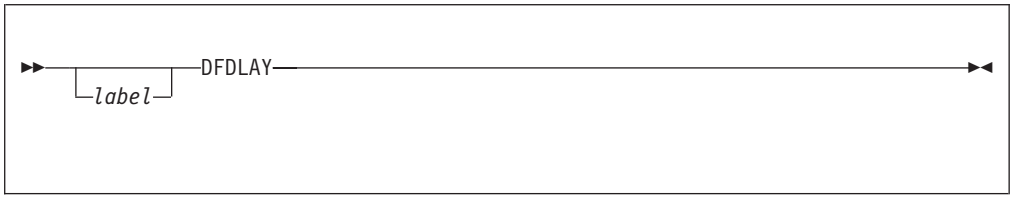
DFDLAY–Delay Processing Conditionally

Use this macro to conditionally delay an entry control block (ECB), forcing it to give up control. The delay occurs if the following is true:

- The ECB is close to an application time-out (system error CTL-000010)
- Symbol &TPFDBDV in the DBLCL macro is set to a non-zero value.

This macro generates specific code for your environment that can be release dependent.

Format



label
is a symbolic name assigned to the macro statement.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

- If symbol &TPFDBDV in the DBLCL macro is set to 0, processing of the ECB will never be delayed. See *TPFDF Installation and Customization* for more information about the DBLCL macro.
- This macro is for use in TPFDF service routines. Do not use the DFDLAY macro in application programs because it can prevent valid CTL-000010 system errors.

Examples

The following example conditionally delays processing of an ECB.

```
DFDLAY
```

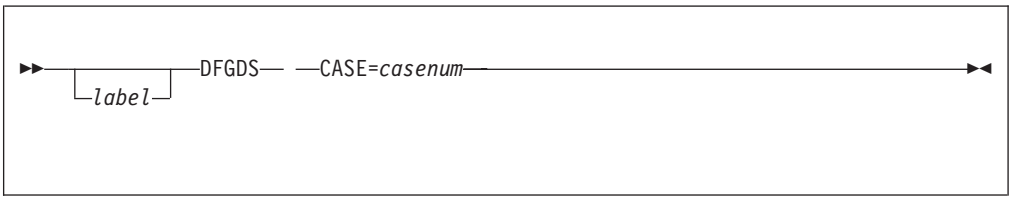
Related Macros

None.

DFGDS–General Data Set Support User Exit

Use this macro to implement general data set support or to obtain the necessary code from a third party.

Format



label
is a symbolic name assigned to the macro statement.

CASE=*casenum*
specifies the case number passed with the macro call, where *casenum* is a decimal number.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

Register values are preserved across this macro call.

Examples

The following example calls GDS case 4.

```
DFGDS CASE=4
```

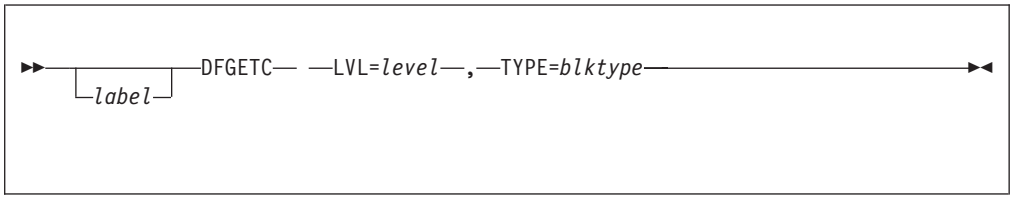
Related Macros

None.

DFGETC–Get Working Storage Block

Use this macro to get a working storage block that can be used by multiple entry control blocks (ECBs). This macro generates specific code for your environment that can be release dependent.

Format



label
is a symbolic name assigned to the macro statement.

LVL=*level*
specifies the data level, where *level* is a free data level from D0–DF.

TYPE=*blktype*
specifies the block type, where:

L0 specifies a 128-byte block size.

L1 specifies a 381-byte block size.

L2 specifies a 1055-byte block size.

L3 specifies a 4000-byte block size. This block type is available only in an ALCS environment.

L4 specifies a 4095-byte block size.

L5 specifies a user-defined size. This block type is available only in an ALCS environment.

L6 specifies a user-defined size. This block type is available only in an ALCS environment.

L7 specifies a user-defined size. This block type is available only in an ALCS environment.

L8 specifies a user-defined size. This block type is available only in an ALCS environment.

Entry Requirements

None.

Normal Return

Register 14 (R14) contains the address of the assigned storage block.

Error Return

None.

Programming Considerations

On return from this macro, the contents of R15 are unknown.

Examples

The following example will get a 1055-byte working storage block on data level 5 that can be shared by multiple ECBs.

```
DFGETC LVL=D5,TYPE=L2
```

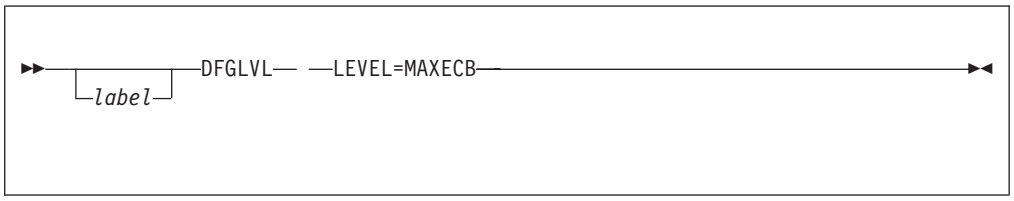
Related Macros

None.

DFGLVL–Get Resource Level

Use this macro to query system resource levels. This macro generates specific code for your environment that can be release dependent.

Format



label
is a symbolic name assigned to the macro statement.

LEVEL=MAXECB
returns the maximum number of entry control blocks (ECBs) that can be active for a TPF system or ALCS environment to process deferred ECBs. If the number of active ECBs exceeds this value, the TPF system or ALCS environment will stop processing deferred ECBs. Deferred ECBs are maintained on the TPFor ALCS defer list.

Entry Requirements

None.

Normal Return

Register 15 (R15) contains the maximum number of ECBs that can be active for a TPF system or an ALCS environment to process deferred ECBs.

Error Return

None.

Programming Considerations

On return from this macro, the contents of R14 are unknown.

Examples

The following example will return the maximum number of ECBs that can be active for the TPF system or ALCS environment to process ECBs on the defer list.

```
DFGLVL LEVEL=MAXECB
```

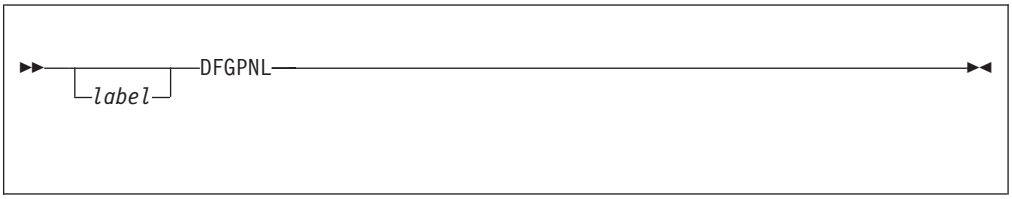
Related Macros

None.

DFGPNL–Get Calling Program Address

Use this macro to get the address of the application program that issued the TPFDF macro or function call. This macro generates specific code for your environment that can be release dependent.

Format



label
is a symbolic name assigned to the macro statement.

Entry Requirements

None.

Normal Return

Register 14 (R14) contains the calling program address.

Error Return

R14 contains 0.

Programming Considerations

On return from this macro, the contents of R15 are unknown.

Examples

The following example returns the address of the application program that issued the TPFDF macro or function call.

```
DFGPNL
```

Related Macros

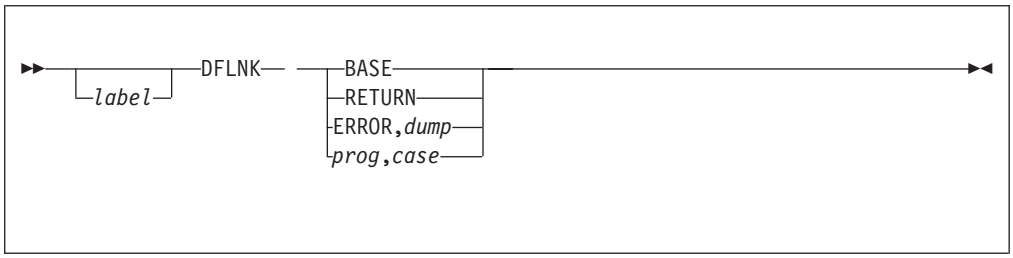
None.

DFLNK–TPFDF Fast Linkage

Use this macro to support TPFDF fast-link segments. Parameters are provided to do the following:

- Call a fast-link segment
- Return from a fast-link segment
- Reestablish the program base
- Call the TPFDF error handler.

Format



label
is a symbolic name assigned to the macro statement.

BASE
reestablishes register (R8) as the program base.

RETURN
specifies that control is returned to the TPFDF macro call.

ERROR
specifies the error code condition.

dump
is the dump number assigned for the central error handler.

prog
is the name of the fast-link segment that gets control.

case
is the case number, which is a decimal number in the range 0–7. The case number is in the segment specified with the *prog* parameter.

Entry Requirements

R1 must point to the current TPFDF stack area.

Normal Return

None.

Error Return

None.

Programming Considerations

If you specify the *prog* parameter value with a UWA*x* segment and the calling segment is not a UW*xx* program, all data levels are preserved and registers in the

range R0–R7 are saved and restored. If the calling segment is a UW*xx* program, an internal fast-link call is used and registers in the range R0–R7, R14, or R15 are not saved.

Examples

- The following example provides a fast link to case number 4 of segment UWB0:

  ```
  DFLNK UWB0,4
  ```

- The following example calls the case for DB0100 in the central error handler:

  ```
  DFLNK ERROR,DB0100
  ```

- The following example issues a GETCC macro on level DD and restores the correct program base:

  ```
  GETCC DD
  DFLNK BASE
  ```

Related Macros

None.

DFSSU–Handling DBDEF Subtables

Use this macro to create a copy of the TPFDF and common subtables in a working storage block. The TPFDF subtable is the portion of the database definition (DBDEF) table that contains TPFDF-specific information such as item length and next available byte (NAB). The common subtable is the portion of the DBDEF table that contains information common to all entry control blocks (ECBs) accessing the file.

The DFSSU macro takes the multiple database function (MDBF) environment into consideration. Any values with subsystem user (SSU) overrides for the specified SSU are modified as appropriate. See *TPFDF Database Administration* for more information about SSU overrides.

Note: The MDBF environment is available only in a TPF system.

Figure 23 shows how the parameters for the DFSSU macro allow access to the copy. The DBDEF parameter points to the base of the DBDEF table to be copied. The BASE parameter specifies which register points to the base of the working storage. The SW02SR DSECT has a set of labels suffixed by the character M to identify fields in the copy area.

Note: SUFFIX=M in SW02SR is reserved for TPFDF product use.

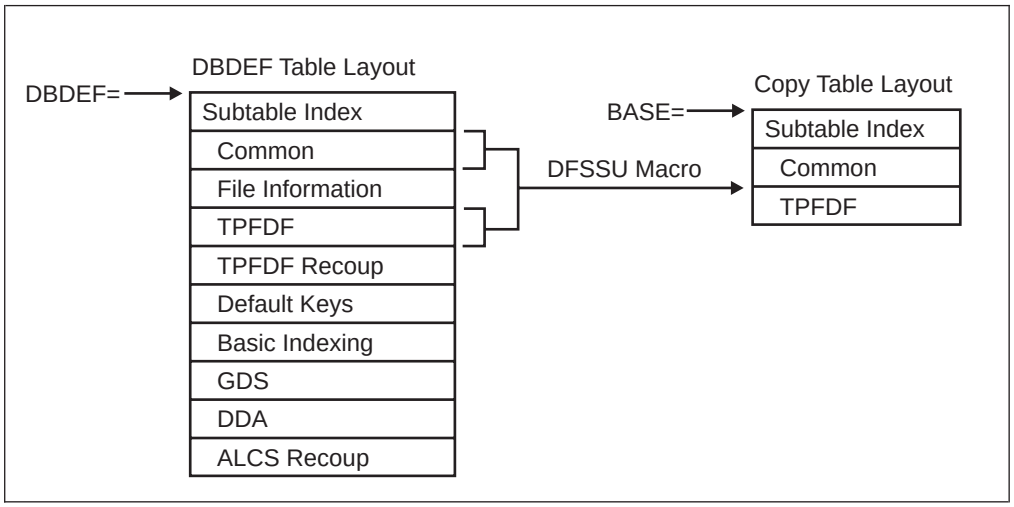
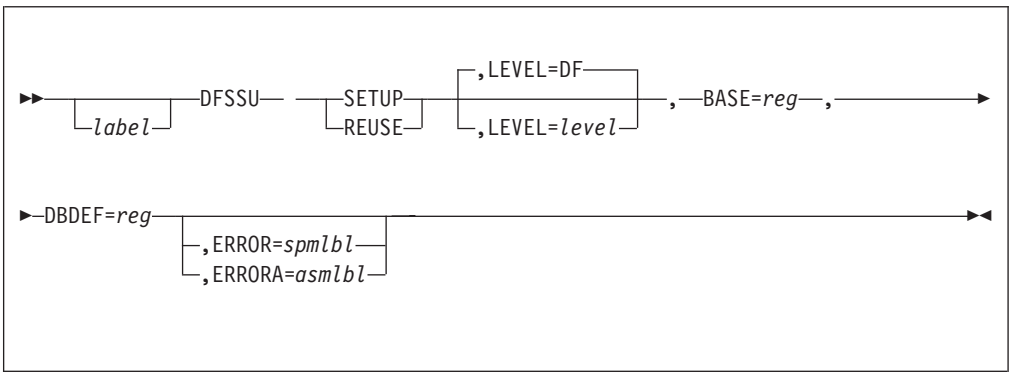


Figure 23. DFSSU Macro: Create a Copy of the TPFDF and COMMON Subtables

Format



label
is a symbolic name assigned to the macro statement.

SETUP
allocates working storage (using a GETCC macro) on the data level specified by the LEVEL parameter.

REUSE
uses previously allocated working storage on the data level specified by the LEVEL parameter.

LEVEL=*level*
specifies the entry control block (ECB) data level, where *level* is the TPF or ALCS ECB data level to which the DBDEF subtables will be copied.

BASE=*reg*
specifies the register that will contain the address of the area to which the DBDEF subtables were copied, where *reg* is R0–R7, R14, or R15.

DBDEF=*reg*
specifies the location of the DBDEF to be copied, where *reg* is a register that points to the base of the DBDEF. Use register 0–7 (R0–R7), R14, or R15 for the value of *reg*.

ERROR=*spmlbl*
specifies where control is returned if the subsystem user of the ECB is excluded from the file whose DBDEF is being copied, where *spmlbl* is a symbolic name defined with the #LOCA structured programming macro (SPM). See *TPFDF and TPF Structured Programming Macros* for more information about SPMs.

ERRORA=*asmlbl*
specifies where control is returned if the subsystem user of the ECB is excluded from the file whose DBDEF is being copied, where *asmlbl* is an assembler label.

Entry Requirements

- If you specify the SETUP parameter, the data level specified by the LEVEL parameter must be free.
- If you specify the REUSE parameter, the data level specified by the LEVEL parameter must refer to a core block that is at least 381 bytes.
- The register that you specify with the DBDEF parameter must contain the address of the DBDEF table to be copied.

Normal Return

- The contents of R0–R7 are preserved across this macro call, except for the register specified with the BASE parameter.
- R14 contains the address of the DBDEF copy.
- The contents of R15 cannot be predicted unless R15 was specified with the BASE parameter.
- The register specified by the BASE parameter contains the address of the DBDEF copy.
- If you specify the SETUP parameter, a 381-byte block containing the DBDEF copy is attached to the data level specified by the LEVEL parameter.
- If you specify the REUSE parameter, the contents of the block on the data level specified by the LEVEL parameter is changed to contain the DBDEF copy.

Error Return

If the subsystem user of the ECB is excluded from the file whose DBDEF is being copied, control is passed to the label specified by the ERROR or ERRORA parameter. If you did not specify the ERROR or ERRORA parameter, control is returned to the next sequential instruction.

Programming Considerations

You can specify the same register for the BASE and DBDEF parameters.

Examples

The following example will make a copy of the DBDEF subtables in the block already attached to data level F (DF). The copy will also be updated with information that is specific to the subsystem user of the ECB. R15 contains the address of the original DBDEF table on entry to the macro and it will contain the address of the DBDEF copy on return from the macro.

```
DFSSU REUSE,LEVEL=DF,DBDEF=R15,BASE=R15
```

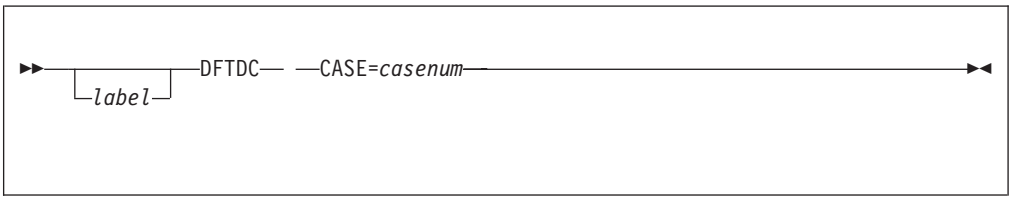
Related Macros

None.

DFTDC–Dialogue Control Facility Support User Exit

Use this macro to implement dialogue control facility support or to obtain the necessary code from a third party.

Format



label
is a symbolic name assigned to the macro statement.

CASE=*casenum*
specifies the case number passed with the macro call, where *casenum* is a decimal number.

Entry Requirements

None.

Normal Return

None.

Error Return

None.

Programming Considerations

None.

Examples

The following example shows how to include case number 5.

```
DFTDC CASE=5
```

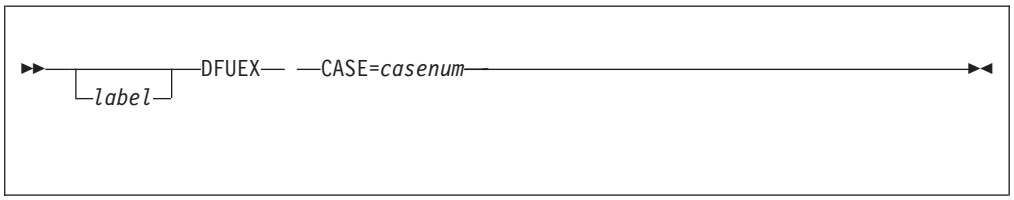
Related Macros

None.

DFUEX–Define TPFDF User Exit Point

Use this macro to define a TPFDF user exit point.

Format



label
is a symbolic name assigned to the macro statement.

CASE=*casenum*
specifies the case number passed with the macro call, where *casenum* is one of the following:

1 initializes customer database definition (DBDEF) tables.

2 excludes multiple database function (MDBF) subsystems from using the TPFDF product.

Entry Requirements

If you specify case 1, register 7 (R7) must contain a return address to subroutine UF00ID in segment UF00.

Normal Return

- If you specify case 1, the contents of R2, R14, and R15 cannot be predicted. All other registers are preserved across this macro call.
- If you specify case 2, symbol &SKPUF00 is set to one of the following:

  0 indicates the subsystem was not excluded.

  1 indicates the subsystem was excluded.

Error Return

None.

Programming Considerations

You can code this macro only in TPFDF segment UF00.

Examples

The following example shows how to include case number 1:

```
DFUEX CASE=1
```

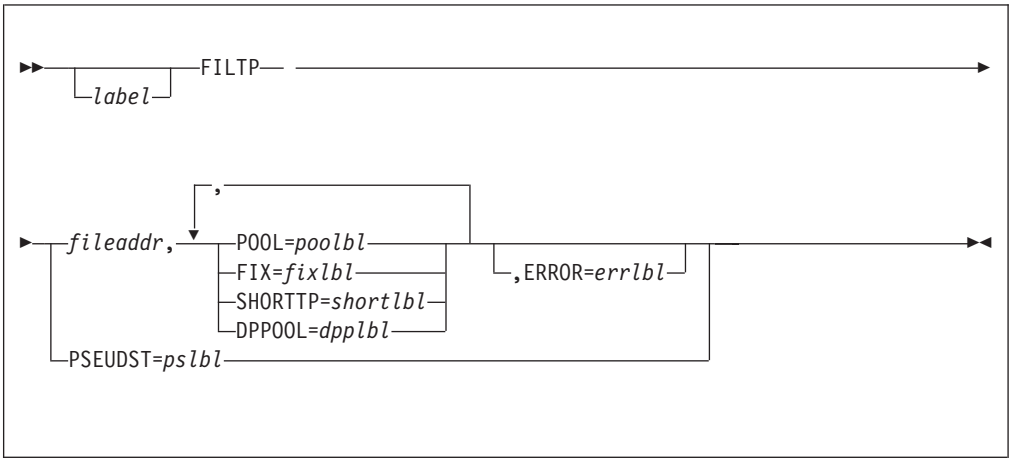
Related Macros

None.

FILTP–Determine File Address Type

Use this macro to determine if a specified file address is a *fixed* file address or a *pool* file address. If the file address is a pool address, it can be further classified as *short-term* or *long-term duplicated*. This macro then branches to a specified label according to the file type that is determined.

Format



label
is a symbolic name assigned to the macro statement.

fileaddr
is the location of a 4-byte field that contains the file address.

POOL=*poolbl*
specifies where to branch if the file address is a pool address, where *poolbl* is the label for that location.

FIX=*fixlbl*
specifies where to branch if the file address is a fixed address, where *fixlbl* is the label for that location.

SHORTTP=*shortlbl*
specifies where to branch if the file address is a short-term pool address, where *shortlbl* is the label for that location.

DPPOOL=*dpplbl*
specifies where to branch if the file address is a long term duplicated pool address, where *dpplbl* is the label for that location.

PSEUDST=*pslbl*
places the dummy file address used for TPFDF W-type files at the label specified by *pslbl*.

ERROR=*errlbl*
specifies where to branch if the specified file address is not valid, where *errlbl* is the label for that location. If you do not specify the ERROR parameter and the specified file address is not valid, control is passed to the next sequential instruction (NSI).

Entry Requirements

When you specify any parameter except the PSEUDST parameter, the file address to be checked must be at the label specified by *fileaddr*.

Normal Return

- When you specify any parameter except the PSEUDST parameter, the contents of register 0 (R0), R14, and R15 cannot be predicted.
- When you specify the PSEUDST parameter, the dummy file address used for TPFDF W-type files is placed at the label specified by *pslbl*.

Error Return

When you specify any parameter except the PSEUDST parameter, control is passed to the label specified by the ERROR parameter if the specified file address is not valid.

Programming Considerations

- If you are running in an ALCS environment, do ***not*** assemble programs using the version of the FILTP macro that is shipped with the TPFDF product. Use the version of the FILTP macro that is provided with the ALCS product.
- The SHORTTP and DPPOOL parameters take precedence over the POOL parameter. That is, if the file address is a short-term pool address, control will be passed to the label specified by *shortlbl*, not the label specified by *poolbl*.

Examples

The following example checks the file address at label EBCFA0.

```
FILTP EBCFA0,SHORTTP=STLABEL,POOL=LTLABEL,DPPOOL=LDLABEL
```

The following shows where control will be passed based on the type of file address:

File Address	Control Passed To
Short-term pool	Label STLABEL
Long-term duplicated pool	Label LDLABEL
Long-term non duplicated pool	Label LTLABEL
Fixed file address	Next sequential instruction
Not valid	Next sequential instruction

Related Macros

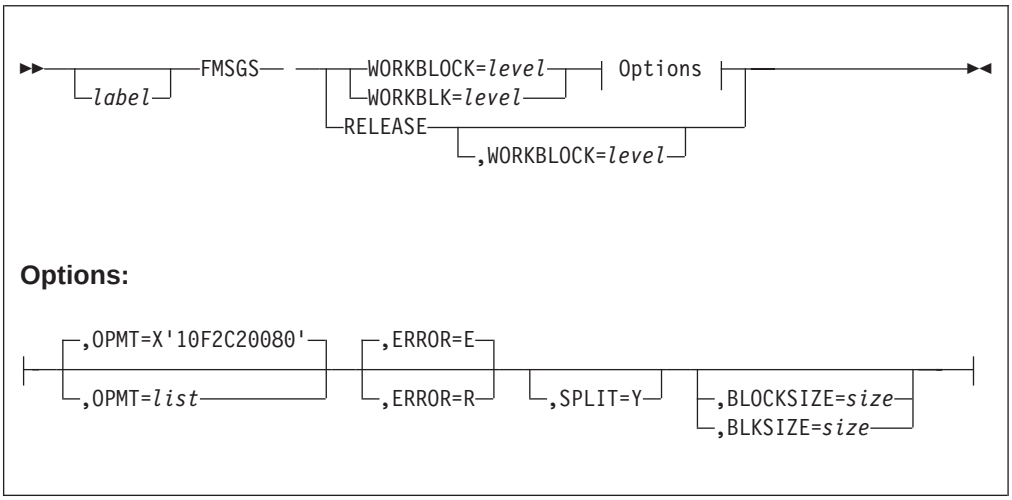
None.

FMSGS–Set Up Output Messages

Use this macro to generate code to set up:

- A work block with output message transmission (OPMT) information
- A blanked-out line area (FMSLNA) of 64 bytes
- The code for activating the TPF or ALCS FMSG program, including:
  - Saving the general registers
  - Calculating the character count (UI2CNF).

Format



label
is a symbolic name assigned to the macro statement.

WORKBLOCK
specifies a data level that is free to set up the work block.

WORKBLK
specifies a data level that is free to set up the work block.

level
is the data level number. Do not use level number 2.

OPMT=*list*
specifies an OPMT parameter (UI2INC–UI2CNN), where *list* is a 5-byte hexadecimal number. The hexadecimal number is used to set indicators in the UI2PF structure starting at UI2INC–UI2CNN.

ERROR
specifies where to return control if an error occurs.

R returns control to the calling program. Register 14 (R14) contains the return code.

E exits the program with a system error.

BLOCKSIZE
specifies the block size.

BLKSIZE
specifies the block size.

size
is one of the following block sizes:

L1 specifies a 381-byte block. This is the default when you do not specify the SPLIT=Y parameter.

L2 specifies a 1055-byte block. This is the default when you specify SPLIT=Y.

SPLIT=Y
generates code to support scrolling. The line number (1–7) is set up in FMSHLN. This code (which also activates FMS3) is activated only if WA0ET1 BIT 2 indicates an RO request (T-ENTRY).

RELEASE
cleans up and releases any storage obtained by previous iterations of the FMSGS macro.

Entry Requirements

- You must activate the subroutines by using the #PERF structured programming macro (SPM). Two subroutines are required:
  – One for FMSGS
  – One for FMSGS RELEASE.

  Note: Register 7 (R7) is used for subroutine linkage.
- The initial call to this subroutine:
  – Sets up the work block on the indicated (and free) level
  – Blanks out the line area
  – Returns to the caller.
- On subsequent entries, R1 must point to the next available byte in the line area (FMSLNA) and R2 must still be the base of OPMT.
- If you specify the SPLIT=Y parameter, you must:
  – Set R2 to point to DSECT UI2PF
  – Set up a branch to UI2PF.

  This is required to allow the header line information to be set up in FMSHLN.
- If a user-supplied *end message* is requested (instead of the standard ENDPARTx message), set up FMSEND and FMSENS.

  The AAA/CAA/MAA/RCB must be on level 1 because the FMSGS macro will interrogate the RO indicator.
- Set the respective bits in UI3CNN to signal that the last line is being submitted.

Normal Return

- The work block is set up and the line area is blanked out.
- R1 points to first available byte of the 64-byte line area (FMSLNA).
- Register 2 (R2):
  – Is the base of OPMT.
  – Is unchanged if you do not specify ERROR=R.
  – Contains minus one (-1) if you specify ERROR=R.
- The contents of R14–R0 and R3–R7 are unchanged unless an error occurs.
- If you specify the RELEASE parameter:
  – Any OMSG (and chains) on level 2 are released and the core blocks are no longer held
  – If you specify the WORKBLOCK parameter, that core block is released on entry to the subroutine.

Error Return

The contents of R2 and R14 are changed if an error occurs. If you specify ERROR=R and an error occurs, R14 contains the error code.

Programming Considerations

- If an exit to UIO is performed, do not use levels 1, 2, 4, 5, or 6 for a work block because the UIO programs use these levels.
- This macro uses approximately 85 to 160 bytes of storage depending on the options selected.

Examples

```
FMSGS WORKBLOCK=0
FMSGS WORKBLOCK=3,OPMT=18F2C80040
FMSGS WORKBLOCK=C,ERROR=R
FMSGS WORKBLOCK=C,SPLIT=Y
FMSGS WORKBLOCK=C,BLKSIZE=L2
FMSGS RELEASE
FMSGS RELEASE,WORKBLOCK=C
```

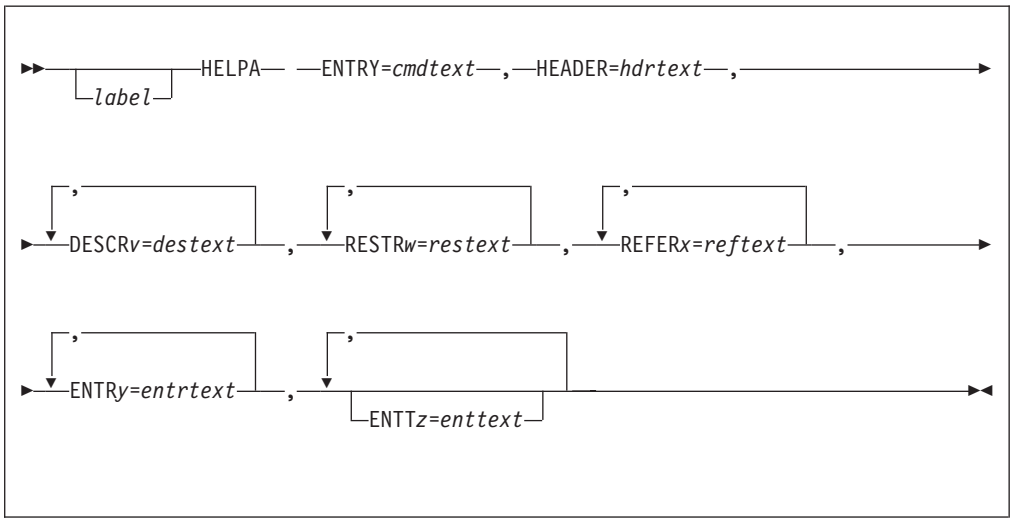
Related Macros

None.

HELPA–Help Message Text

Use this macro to build the output messages for the ZUDFM OAH (help) command.

Format



label
is a symbolic name assigned to the macro statement.

ENTRY=*cmdtext*
specifies the command for which you want to build the help message, where *cmdtext* is a 2- to 10-character string.

HEADER=*hdrtext*
specifies the header for the help message, where *hdrtext* is a 1- to 30-character string.

DESCR*v***=***destext*
specifies the command description that will be displayed in the help message, where *v* is a number from 1–7 and *destext* is a 1- to 45-character string.

RESTR*w***=***restext*
specifies any restrictions that will be displayed in the help message, where *w* is a number from 1–5 and *restext* is a 1- to 45-character string.

REFER*x***=***reftext*
specifies any references that will be displayed in the help message, where *x* is 1 or 2 and *reftext* is a 1- to 45-character string.

ENTR*y***=***entrtext*
specifies the different parameters that will be displayed in the help message, where *y* is a number from 1–20 and *entrtext* is a 1- to 25-character string.

ENTT*z***=***enttext*
specifies the description of the parameters specified with the ENTR*y* parameter, where *z* is a number from 1–20 and *entrtext* is a 1- to 45-character string.

Entry Requirements

None.

Normal Return

The help message is displayed.

Error Return

None.

Programming Considerations

- Register values are preserved across this macro call.
- You must code all spaces in the text strings as an underscore character (_).
- The lines of text are displayed in sequential order based on the parameters. That is, the text defined for DESCR1 displays first, followed by the text for DESCR2, and so on. For example, to display the following:

```
FIRST LINE OF TEXT
SECOND LINE OF TEXT
THIRD LINE OF TEXT
```

  code the parameters as follows:

```
DESCR1=FIRST_LINE_OF_TEXT,
DESCR2=SECOND_LINE_OF_TEXT,
DESCR3=THIRD_LINE_OF_TEXT
```

Examples

The following example shows how to code a help message for the ZUDFM OAI/FILE command.

```
HELPA ENTRY=OAI/FILE,                                        -
      HEADER=DSECT_LABEL_DISPLAY,                            -
      DESCR1=THE_DSECT_EQUATES_AND_LABELS_WRITTEN,           -
      DESCR2=ONTO_MLS_BY_OFFLINE_PGMS_CAN_BE_DISPLAYED,      -
      DESCR3=ONLINE_WITH_OAI/FILE_ENTRIES,                   -
      RESTR1=NONE,                                           -
      REFER1=SEE_UFB_DOCUMENTATION,                          -
      ENTR01=OAI/FILE_,                                      -
      ENTT01=INDEX_OF_ALL_DSECTS_WRITTEN_ONTO_MLS,           -
      ENTR02=OAI/FILE/ID,                                    -
      ENTT02=DISPLAY_DETAILED_INFORMATION_OF_SPECIFIED_ID.,  -
      ENTT03=LREC-KEYS_VALIDATION/EQUATES/LABELS_WITH,       -
      ENTT04=DISPLACEMENT_AND_LENGTH
```

This example produces the following display:

```
User:   ZUDFM OAHOAI/FILE

System: UDFM0243I          TPFDF HELP FACILITY
        O A - H E L P  -  OAI/FILE    DSECT LABEL DISPLAY
        ------------------------------------------------------------

        DESCRIPTION:      THE DSECT EQUATES AND LABELS WRITTEN
                          ONTO MLS BY OFFLINE PGMS CAN BE DISPLAYED
                          ONLINE WITH OAI/FILE ENTRIES
        RESTRICTION:      NONE
        REFERENCE:        SEE UFB DOCUMENTATION

        ENTRIES:
        OAI/FILE          INDEX OF ALL DSECTS WRITTEN ONTO MLS
        OAI/FILE/ID       DISPLAY DETAILED INFORMATION OF SPECIFIED ID.
                          LREC-KEYS VALIDATION/EQUATES/LABELS WITH
                          DISPLACEMENT AND LENGTH
```

Related Macros

None.

Index

Special Characters

DF_EF function 14
DF_EMPTY function 14
DF_ER function 14
DF_ERBTR function 14
DF_ERCNT function 14
DF_ERDSP function 14
DF_ERLST function 14
DF_ERX function 14
df_nbrkeys function 124
DF_NR function 14
DF_OK function 14
DF_SERRC function 14
df_setkey_bool function 150
df_setkey_dbdef function 150
df_setkey_mod function 150
df_setkey function 150
DF_TEST function 14
dfadd function 73
dfadr function 80
dfckp function 83
dfclr function 85
dfcls function 86
dfcpy function 92
dfcre function 95
dfdel function 97
dfdix function 105
dfdsp function 107
dffrl function 111
dfidx function 112
dfifb function 114
dfkey function 115
dfmod function 117
dfmrg function 121
dfopn function 125
dfopt function 130
dfred function 134
dfrep function 143
dfret function 145
dfrst function 147
dfspa function 156
dfsrt function 157
dftab function 170
dftld function 160
dftlg function 163
dftrd function 166
dfuky function 167
member_size function 168

A

activating a key list 115, 247
adding
  dfadd function 73
  DBADD macro 176
  extended LRECs 73, 176
  LRECs 73, 176
adding *(continued)*
  subLRECs 73, 176
  userLRECs 73, 176
algorithm argument
  specifying 4
application program
  sample 33
    assembler 39
    C language 48
application programming
  overview 3

B

begin transaction 7
BLKSZ macro 328
block header
  dfred function 134
  DBRED macro 272
  reading 134, 272
block type
  BLKSZ macro 328
  converting 328
Boolean logic
  example of using 141, 284
  using in key lists 30

C

C language interface
  DFCLIB macro 338
calculating
  member_size function 168
  DBCNT macro 332
  set symbol length 332
  size of a member 168
checkpointing
  dfckp function 83
  DBCKP macro 195
  open subfiles 83, 195
closing
  dfclr function 85
  dfcls function 86
  DBCLR macro 198
  DBCLS macro 199
  subfile 86, 199
  subfile without a dump 85, 198
commit scopes
  benefits 10
  checkpoint processing 10
  close processing 10
  examples 9
  internal uses 10
  nested 8
  programming conventions 8
  root 8
  suspended 8

© Copyright IBM Corp. 1997, 2001

commit transaction 7
conventions for transaction manager (TM) 8
copying
  dfcpy function 92
  DBCPY macro 205
  subfile 92, 205
creating
  dfcre function 95
  dfidx function 112
  dfopn function 125
  dfspa function 156
  database interface block (DBIFB) 125, 260
  DBCRE macro 210
  DBIDX macro 241
  DBOPN macro 260
  DBSPA macro 304
  detail file 95, 210
  empty subfile 95, 210
  index reference 112, 241
  pool subfile 95, 210
  work space 125, 156, 260, 304
current LREC 77, 186

D

DASD
  dfckp function 83
  dfcls function 86
  dftld function 160
  DBCKP macro 195
  DBCLS macro 199
  DBTLD macro 313
  writing a subfile to 160, 313
  writing blocks to 83, 86, 195, 199
data level independence (DLI)
  dfopn function 129
  DBOPN macro 271
  specifying 129, 271
data levels 3
database definition (DBDEF) table
  dftab function 170
  accessing 170, 333
  DBTAB macro 333
  DFUEX macro 354
  user-specific entries 354
database interface block (DBIFB)
  dfifb function 114
  dfopn function 125
  checking 114, 244
  creating 125, 260
  DBIFB macro 244
  DBOPN macro 260
DBADD macro 176
DBADR macro 190
DBCKP macro 195
DBCLR macro 198
DBCLS macro 199
DBCNT macro 332
DBCPY macro 205
DBCRE macro 210
DBDEL macro 214
DBDIX macro 228
DBDSP macro 231
DBEMPTY parameter 14
DBEOF parameter 14
DBERROR parameter 14
DBFOUND parameter 14
DBFRL macro 240
DBIDX macro 241
DBIDX parameter 14
DBIFB macro 244
DBKEY macro 247
DBMOD macro 249
DBMRG macro 254
DBOPN macro 260
DBRED macro 272
DBREP macro 286
DBRET macro 290
DBRST macro 293
DBSETK macro 298
DBSPA macro 304
DBSRT macro 307
DBTAB macro 333
DBTLD macro 313
DBTLG macro 318
DBTRD macro 323
DBUKY macro 325
default keys
  example of reading with 141, 284
  specifying LRECs with 19
default-key key list
  df_setkey_dbdef function 150
  DBSETK macro 298
  definition of 26
  example of using 141, 284
  setting up a key 150, 298
  using 31
deleting
  dfdel function 97
  dfdix function 105
  DBDEL macro 214
  DBDIX macro 228
  extended LRECs 97, 214
  index reference 105, 228
  LRECs 97, 214
  subLRECs 97, 214
detac mode
  dfckp function 83
  dfcls function 86
  dfopn function 127
  committing updates to DASD 83, 86, 195, 199
  DBCKP macro 195
  DBCLS macro 199
  DBOPN macro 264
  discarding updates 86, 199
  opening a subfile in 127, 264
  placing a subfile in 83, 195
  taking a subfile out of 83, 195
detail file
  dfcre function 95
  creating 95, 210
  DBCRE macro 210

DFCAS macro 336
DFCLIB macro 338
DFDDA macro 341
DFDLAY macro 342
DFGDS macro 343
DFGETC macro 344
DFGLVL macro 346
DFGPNL macro 347
DFLNK macro 348
DFSSU macro 350
DFTDC macro 353
DFUEX macro 354
diagrams for macro models xiv
dialogue control facility 353
displaying
  dfdsp function 107
  DBDSP macro 231
  LRECs 107, 231

E

entry control block (ECB)
  dffrl function 111
  DBFRL macro 240
  free a level 111, 240
errors
  checking
    using C functions 14
    using equates 14
    using structured programming macros (SPMs) 14
  checking SW00RT1 13
  checking SW00RT2 13
  checking SW00RTN 13
  identifying 13
  type of 13
extended LREC
  dfadd function 73
  dfdel function 97
  dfred function 134
  dfrep function 143
  adding 73, 176
  DBADD macro 176
  DBDEL macro 214
  DBRED macro 272
  DBREP macro 286
  deleting 97, 214
  reading 134, 272
  replacing 143, 286
  subLREC
    adding 73, 176
    deleting 97, 214
    how TPFDF product adds 4
    how TPFDF product numbers 4
    replacing 143, 286
  userLREC
    adding 73, 176
    replacing 143, 286
  using 4

F

fast-link segments
  DFCAS macro 336
  DFLNK macro 348
  supporting 336, 348
file address type
  determining 355
  FILTP macro 355
file organization
  rules for 23
  specifying with keys 23
FILTP macro 355
finding base address of a SW00SR 114, 244
FMSGS macro 357
free an ECB data level 111, 240
fullfile processing
  dfadr function 80
  dfdel function 101
  dfdsp function 108
  dfmod function 118
  dfmrg function 121
  dfred function 138
  dfsrt function 157
  dftlg function 164
  DBADR macro 190
  DBDEL macro 219
  DBDSP macro 233
  DBMOD macro 251
  DBMRG macro 255
  DBRED macro 277
  DBSRT macro 309
  DBTLG macro 320
  deleting LRECs 101, 219
  displaying LRECs 108, 233
  merging LRECs 121, 255
  modifying LRECs 118, 251
  reading LRECs 138, 277
  sorting LRECs 157, 309
  specifying range of ordinals 80, 190
  writing LRECs to tape 164, 320

G

general C functions
  DF_EF function 14
  DF_ER function 14
  DF_ERBTR function 14
  DF_ERCNT function 14
  DF_ERDSP function 14
  DF_ERLST function 14
  DF_ERX function 14
  df_nbrkeys function 124
  DF_NR function 14
  DF_OK function 14
  DF_SERRC function 14
  df_setkey function 150
  DF_TEST function 14
  dfadd function 73
  dfadr function 80
  dfckp function 83

general C functions *(continued)*
dfclr function 85
dfcls function 86
dfcpy function 92
dfcre function 95
dfdel function 97
dfdix function 105
dfdsp function 107
dffrl function 111
dfidx function 112
dfifb function 114
dfkey function 115
dfmod function 117
dfmrg function 121
dfopn function 125
dfopt function 130
dfred function 134
dfrep function 143
dfret function 145
dfrst function 147
dfspa function 156
dfsrt function 157
dftld function 160
dftlg function 163
dftrd function 166
dfuky function 167
member_size function 168
checking for errors with 14
using basic indexing with 5
general data set support
DFGDS macro 343
implementing 343
general macros
DBADD macro 176
DBADR macro 190
DBCKP macro 195
DBCLR macro 198
DBCLS macro 199
DBCPY macro 205
DBCRE macro 210
DBDEL macro 214
DBDIX macro 228
DBDSP macro 231
DBFRL macro 240
DBIDX macro 241
DBIFB macro 244
DBKEY macro 247
DBMOD macro 249
DBMRG macro 254
DBOPN macro 260
DBRED macro 272
DBREP macro 286
DBRET macro 290
DBRST macro 293
DBSETK macro 298
DBSPA macro 304
DBSRT macro 307
DBTLD macro 313
DBTLG macro 318
DBTRD macro 323
DBUKY macro 325
general macros *(continued)*
using basic indexing with 5
generating a unique key 167, 325
global modification
df_setkey_mod function 150
dfmod function 117
DBMOD macro 249
DBSETK macro 298
example of using 120, 252
modification key list 26, 31
performing 31, 117, 249

H

HELPA macro 360

I

index reference
dfdix function 105
dfidx function 112
creating 112, 241
DBDIX macro 228
DBIDX macro 241
deleting 105, 228
index support
adding LRECs using 5
reading an LREC using 5
using basic indexing 5
using keys 5

K

key list
df_nbrkeys function 124
df_setkey_bool function 150
df_setkey_dbdef function 150
df_setkey_mod function 150
df_setkey function 150
dfkey function 115
activating 115, 247
DBADD macro 176
DBDEL macro 214
DBKEY macro 247
DBMRG macro 254
DBRED macro 272
DBSETK macro 298
DBSRT macro 307
example of using 27
setting up 26
setting up a key 150, 298
setting up the number of keys 124
specifying LRECs with 19
types 26
default-key 31
modification 31
using 26
using Boolean logic in 30
KEY*n* parameters
DBADD macro 176
DBDEL macro 214

KEY*n* parameters *(continued)*
DBDSP macro 231
DBMRG macro 254
DBOPN macro 260
DBRED macro 272
DBSRT macro 307
file organization 23
specifying LRECs with 19
keys
activating 19
Boolean logic 30
default-key key list 31
file organization 23
modification key list 31
partial keys 20
specifying LRECs with 19
using a key list 26
using when opening subfile 20
using with a B+Tree index 21
variable-length fields 22

L

logical record (LREC)
`dfadd` function 73
`dfdel` function 97
`dfdsp` function 107
`dfmod` function 117
`dfmrg` function 121
`dfred` function 134
`dfrep` function 143
`dfret` function 145
`dfsrt` function 157
adding 73, 176
current 77, 186
DBADD macro 176
DBDEL macro 214
DBDSP macro 231
DBMOD macro 249
DBMRG macro 254
DBRED macro 272
DBREP macro 286
DBRET macro 290
DBSRT macro 307
deleting 97, 214
displaying 107, 231
global modification of 31
grouping together 6
indicating a change to 117, 249
merging 121, 254
reading 134, 272
using a B+Tree index 21
using keys 20
using partial keys 20
replacing 143, 286
retaining current address 145, 290
sorting 157, 307
specifying with keys 19
using unique keys 6
using variable-length key 22

M

macro model diagrams xiv
merging
`dfmrg` function 121
DBMRG function 254
LRECs 121, 254
subfiles 121, 254
models of macro invocations xiv
modification key list
`df_setkey_mod` function 150
DBSETK macro 298
definition of 26
example of using 120, 252
setting up a key 150, 298
using 31
modifying
`dfmod` function 117
DBMOD macro 249
LRECs 117, 249

N

nested commit scope 8

O

opening
`dfopn` function 125
DBOPN macro 260
subfile 125, 260
ordinal number
`dfadr` function 80
DBADR macro 190
specifying range 80, 190

P

pool subfile
`dfcre` function 95
creating 95, 210
DBCRE macro 210
prime block
`dfadr` function 80
DBADR macro 190
getting file address of 80, 190
program address
DFGPNL macro 347
getting 347
programming conventions for transaction manager (TM) 8

R

railroad tracks xiv
reading
`dfred` function 134
`dftrd` function 166
block header 134, 272
DBRED macro 272
DBTRD macro 323

reading *(continued)*
extended LRECs 134, 272
LRECs 134, 272
subfiles from tape 166, 323
replacing
dfrep function 143
DBREP macro 286
extended LRECs 143, 286
LRECs 143, 286
subLRECs 143, 286
userLRECs 143, 286
resource level
DFGLVL macro 346
getting 346
restoring
dfrst function 147
DBRST macro 293
subfile 147, 293
restricted C functions
dftab function 170
restricted macros
BLKSZ macro 328
DBCNT macro 332
DBTAB macro 333
DFCAS macro 336
DFCLIB macro 338
DFDDA macro 341
DFDLAY macro 342
DFGDS macro 343
DFGETC macro 344
DFGLVL macro 346
DFGPNL macro 347
DFLNK macro 348
DFSSU macro 350
DFTDC macro 353
DFUEX macro 354
FILTP macro 355
FMSGS macro 357
HELPA macro 360
resume transaction 7
retaining
dfret function 145
DBRET macro 290
LREC address 145, 290
rollback transaction 7
root commit scope 8
rules for using keys 19

S

sample application program 33
assembler 39
C language 48
solution using basic indexing 33
selection key list
df_nbrkeys function 124
df_setkey function 150
DBSETK macro 298
definition of 26
setting up a key 150, 298
sort/merge key list
df_setkey function 150
dfmrg function 121
dfsrt function 157
DBMRG macro 254
DBSETK macro 298
DBSRT macro 307
definition of 26
setting up a key 150, 298
sorting
dfsrt function 157
DBSRT macro 307
LRECs 157, 307
subfiles 157, 307
structure
member_size function 168
calculating size of member 168
structured programming macros (SPMs)
checking for errors 14
subfile
dfckp function 83
dfclr function 85
dfcls function 86
dfcpy function 92
dfcre function 95
dfdix function 105
dfdsp function 107
dfmrg function 121
dfopn function 125
dfopt function 130
dfrst function 147
dfsrt function 157
dftld function 160
dftlg function 163
dftrd function 166
checkpointing 83, 195
closing 86, 199
closing without a dump 85, 198
copying 92, 205
creating 95, 210
DBCKP macro 195
DBCLR macro 198
DBCLS macro 199
DBCPY macro 205
DBCRE macro 210
DBDIX macro 228
DBDSP macro 231
DBMRG macro 254
DBOPN macro 260
DBRST macro 293
DBSRT macro 307
DBTLD macro 313
DBTLG macro 318
DBTRD macro 323
deleting an index reference to 105, 228
displaying LRECs from 107, 231
identifying 3, 4
merging LRECs in 121, 254
opening 125, 260
opening in detac mode 127, 264
placing in detac mode 83, 195

subfile *(continued)*
reading from tape 166, 323
restoring 147, 293
setting options after opening 130
sorting LRECs in 157, 307
writing to DASD 160, 313
writing to tape 163, 318
suspend transaction 7
suspended commit scope 8
SW00RT1 13
SW00RT2 13
SW00RTN 13
SW00SR
`dfifb` function 114
DBIFB macro 244
finding base address 114, 244
syntax diagrams xiv

T

tape
`dftlg` function 163
`dftrd` function 166
DBTLG macro 318
DBTRD macro 323
reading a subfile from 166, 323
writing a file or subfile to 163, 318
TPFDF Distributed Data Access (TPFDF/DDA)
DFDDA macro 341
implementing 341
transaction manager (TM)
ALCS support 8
commit scopes 8
benefits 10
checkpoint processing 10
close processing 10
examples 9
internal uses 10
nested 8
programming conventions 8
root 8
suspended 8
overview 7
transactions 7
begin 7
commit 7
resume 7
rollback 7
suspend 7

U

unique key
`dfuky` function 167
DBUKY macro 325
generating 167, 325
using 6
user exit point 354

W

work space
`dfopn` function 125
`dfspa` function 156
creating 125, 156, 260, 304
DBOPN macro 260
DBSPA macro 304
working storage block
DFGETC macro 344
getting 344
writing
`dfckp` function 83
`dfcls` function 86
`dftld` function 160
`dftlg` function 163
blocks to DASD 83, 86, 195, 199
DBCKP macro 195
DBCLS macro 199
DBTLD macro 313
DBTLG macro 318
files to tape 163, 318
subfiles to DASD 160, 313
subfiles to tape 163, 318

File Number: S370/30XX-40
Program Number: 5706-196

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH31-0179-08