

**zTPFDF**

**Concepts & Programming**

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Title: Soft Product ™ or ®  Name of manual (e.g., SoftLab® v4.4 System Design I) | | | | |
| ID: |  | | | |
| Effective: | Month day, year | | | |
| Edition No. | Release Date | Author’s Name | Software Version | Reason for Change / Description of Changes |
| 1 | Mm/dd/yyyy |  |  | New training manual for new software version |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Table of Contents**

1. INTRODUCTION TO zTPFDF 8

1.1. INTRODUCTION TO ZTPFDF 9

1.2. DIFFERENCE BETWEEN TPF AND TPFDF DATABASE ACCESS 9

1.3. ADVANTAGES AND DISADVANTAGES OF ZTPFDF 10

1.4. TPFDF COMPONENTS 11

2. TPFDF DEFINITIONS AND BASIC TERMS 13

2.1 TPDFDF DEFINITIONS. 14

2.1.1 TPFDF FILE: 14

2.1.2 SUB-FILE: 14

2.1.3 BLOCKS 14

2.1.4 LOGICAL RECORDS (LRECS): 14

2.1.5 STANDARD TPFDF HEADER 14

2.2 TYPES OF LRECS 15

2.2.1 FIXED-LENGTH LREC 16

2.2.2 VARIABLE-LENGTH LREC 16

2.3 ALGORITHM 16

2.4 DATA LEVEL USAGE 16

2.5 DEFINING TPFDF FILES AND LREC'S 17

2.6 NAME CONVENTION FOR TPFDF FILE NAMES 17

2.7 DBDEF 18

2.8 ORGANISATION OF LRECs 21

2.9 TPFDF FILE HANDLING 21

2.10 FIELDS OF SW00SR THAT PROGRAMMER SHOULD KNOW 22

3. TPFDF FILE COMMANDS AND COMMAND SEQUENCE 23

3.1 FILE COMMANDS 24

3.2 COMMANDS SEQUENCE 24

4. TPFDF – GENERAL MACROS 25

4.1 CODING A TPFDF COMMAND MACRO 26

4.2 COMMON PARAMETERS USED IN TPFDF COMMAND MACROS 26

4.2.1 "REF =" Parameter: 26

4.2.2 FILE =" Parameter: 26

4.2.3 "SUFFIX =" Parameter: 26

4.3 PARAMETERS TO PICK ONE SUB-FILE 27

4.4 PARAMETERS USED TO SPECIFY LREC (S) TO OPERATE 27

4.4.1 DEFINITIONS 27

4.4.2 "MATCHING" 27

4.4.3 ORGANIZATION 27

4.4.4 BASIC POINTS RELATED TO THE KEY PARAMETERS 28

4.4.5 Sub-parameter used with KEYn Parameter 29

*5.* TPFDF MACROS: 31

6. OPEN A SUB-FILE - DBOPN 31

7. dfopn–OPEN A SUB-FILE 32

8. TPFDF Macros: Read an LREC -- DBRED 34

9. dfred–Read one or more logical records 36

10. TPFDF Macros: Close a Sub-file(s) -- DBCLS 38

11. dfcls–Close a sub file 39

12. CHECKING RETURN CONDITIONS OF TPFDF COMMAND MACROS 41

13. ADD an LREC -- DBADD 42

14. dfadd–Add one or more logical records to a sub-file 44

15. DELETE AN LREC -- DBDEL 46

16. dfdel–Delete one or more logical records 48

17. TPFDF MACROS: MODIFY AN LREC -- DBMOD 49

18. dfmod–Perform or indicate logical record modifications 50

19. RETAIN An LREC--- DBRET 51

20. TPFDF Macros: Replace an LREC -- DBREP 51

21. dfrep–Replace a logical record with another logical record 53

22. TPFDF MACROS: MISCELLANEOUS GENERAL MACROS 53

23. DBCRE COMMAND MACRO: 54

24. dfcre–Create a sub-file 55

25. The DBADR Command Macro: 55

26. dfadr–Provide the file address of a prime block 57

27. The DBDSP Command Macro: 58

28. dfdsp–Display logical records from a sub-file 59

29. The DBKEY Command Macro: 61

30. dfkey–Activate a key list 62

31. The DBSETK Command Macro: 63

32. df\_setkey–Setting up a key in a key list 65

33. The DBSPA Command Macro: 68

34. dfspa–Create work space 69

35. The DBIFB Command Macro : 70

36. dfifb–Check a SW00SR slot 71

37. The DBFRL Command Macro : 72

38. dffrl–Ensure an ECB data level is free 72

39. The DBCKP Command Macro : 74

40. dfckp–Checkpoint a sub-file 75

41. The DBCLR Command Macro : 76

4. TPFDF ADVANCED MACRO: 76

1. Copy a sub-file (DBCPY) 76

2. dfcpy–COPY A SUB-FILE 77

3. DBRST: Restore a sub-file 79

4. dfrst–RESTORE A SUB-FILE 81

Return conditions 81

Examples 82

5. DBSRT: Sort a file 83

6. dfsrt–SORT A SUB FILE 85

Examples 86

7. DBMRG: Merge two files 86

Examples 88

9. TPFDF Indexing Support 89

10. BLOCK INDEXING 89

11. B+TREE Indexing 90

13. TPFDF Macros : Create an Index -- DBIDX 92

Format 93

Examples 94

TPFDF Macros : Delete an Index -- DBDIX 94

Format 95

Examples 96

TPFDF Partitioning and Interleaving 97

Using UNIQUE key 99

ADD CURRENT Files 100

Push down chaining 101

TPFDF Option Byte Settings 102

1.1.1. TPFDF Option Byte Settings 102

1.1.2. TPFDF Option Byte Settings 104

14. TPFDF MACRO DESIGN 105

5. TPFDF MACRO DESIGN - Contents 107

1. Physical Design – Guidelines 108

2. Physical Design – Guidelines 109

3. Indexing 111

4. Coding DSECT 111

Initial Area & Global Set Symbols 113

Meaning of the Global Indicators 114

5. Block Header 116

i. LREC definition 117

ii. User Fields 118

Algorithm 119

Default Keys 120

Indexing Parameters 121

iii. Guidelines – Logical Records 122

iv. Block Sizes 123

v. Block Sizes 124

Tips on using Indexing 125

6. TPFDF UTILITIES 127

TPFDF Utilities – Contents 129

CRUISE – Introduction 130

TPFDF Online Maintenance – ZUDFM Commands 131

OAI(ACCESS) – File access using ID or DSECT name 132

OAI/DBTAB(DEF) – Display DBDEF table items 134

i. 134

OAINIT(INIT)– Initialize files 137

OA(FAD) – Display the contents of a sub-file 138

OA\*(DISPLAY) – Display LRECs in a sub-file 139

OAA(ADD) – Add LRECs to a sub-file 142

OAX(DELETE) – Delete LRECs from a sub-file 144

OAR(REPLACE) – Replace User data in an LREC 145

OAF(FCH) – Display Forward & Backward chaining 147

OAP(PACK) – Pack a sub-file 148

OAL(LOG) – Display recent OA commands 149

OAE(RESTRICT) – Restrict Table Handling 150

OAZ(TOD) – Convert time 151

7. tpfdf quick reference tables 152

TPFDF Basic Dumps 153

i. Table of TPFDF Algorithms 154

SW00RTN Conditions returned from TPFDF Commands 156

TPFDF Option Byte Settings 157

Meaning of the Global Indicators 158

ZUDFM Entries for Online Maintenance of TPFDF Files 159

# INTRODUCTION TO zTPFDF

This chapter will provide introduction to zTPFDF.

Following topics will be discussed in this section

* Introduction to zTPFDF
* Difference between zTPF and zTPFDF Database Access
* Advantage and Disadvantage of zTPFDF
* zTPFDF Components

Following zTPFDF components are discussed in this section.

* DBDEF Table, DBDEF Macros and DSECT Marcos
* DBIFB Block
* Macros and Functions
* Maintenance and Test Utility (ZUDFM)
* Data Collection Utility (ZUDFC)
* Capture/Restore utility, information and statistics environment (ZFCRU)
* Recoup Utility

## INTRODUCTION TO z/TPFDF

* z/TPFDF is a database manager for application programs that run in a z/TPF operating environment.
* z/TPFDF product increases programmer productivity by providing centralized database routines so that application programmers need to understand only the logical relationships of data, not the physical characteristics.
* z/TPFDF product helps to simplify the job of the application programmer by allowing high-speed access to persistent data on the z/TPF system while providing a simple application interface.
* z/TPFDF is a standardized interface between the z/TPF applications and the online database.
* z/TPFDF acts as a middleware database management system for z/TPF files, and works as a bridge between Application Programs and z/TPF by providing Application interface macros.

## DIFFERENCE BETWEEN z/TPF AND z/TPFDF DATABASE ACCESS

**TRADITIONAL z/TPF DATABASE ACCESS**

ENTRC FACE

FIWHC D4,label

Setup FCH Address

RCUNC D4

FINWHC D4,label

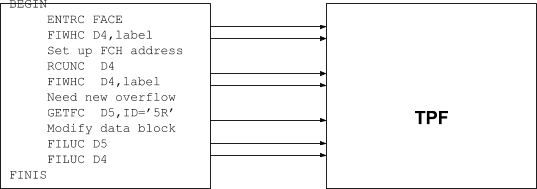
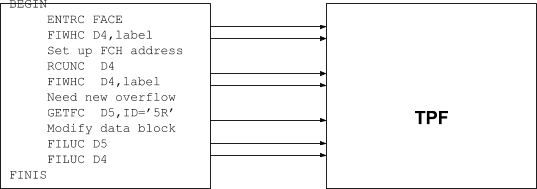
Need new overflow

GETFC D5,ID=’5R’

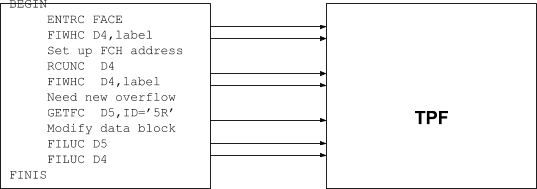
Modify data block

FlLUC D5

FILUC D5

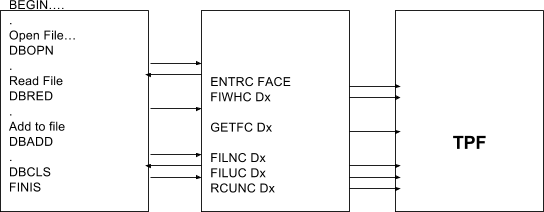


z/TPF



* There is no Database Manager
* Each Application performs own database maintenance.
* Programmer needs to be aware of size and location of the data
* Same routines are coded multiple times by multiple applications.

**z/TPFDF DATABASE ACCESS**



Application Program z/TPFDF Program Control Program

* In the z/TPFDF environment
  + Application deals with logical records
  + The physical aspects of accessing a logical record are all handled by z/TPFDF program called through functions.
  + The application does not know on what levels the physical records were retrieved or how many chain records exist.
  + Even though z/TPFDF deals with the physical issues, the application still is very much aware of the physical data organization, and in most cases, applications logic is based on that knowledge
  + There is some extra CPU utilization time compared to developing a similar database using the traditional z/TPF database environment, but no extra I/O.
  + There are many z/TPFDF database design options available to optimize performance.

## ADVANTAGES AND DISADVANTAGES OF z/TPFDF

* **ADVANTAGES**
* z/TPF Programmers spend 80% of the time in taking care of handling the database access and maintenance and 20% of the time focusing on the business logic.
* Usage of z/TPFDF would help the programmer forget about database issues such as
  + Ensuring the Database Chaining
  + Ensuring the Integrity of the Database
  + Making sure space utilization is optimal in the database etc.
  + z/TPFDF Programmer is not worried about the Physical size, Pool Type and physical location of data within the file.
* z/TPFDF provides standard interface macros to perform read, modify, update, add, delete and other logical operations on z/TPF Files
* Data is referred logically as a ‘Logical Record’ with certain key attributes, as opposed to a physical location based identification in traditional file handling
* Usage of z/TPFDF brings about a large increase in programmer productivity due to
  + Rapid Application Development
  + Better Application Maintenance
  + Features such as Online utilities to maintain the database and CRUISE offer excellent capabilities to maintain and optimize the performance of TPFDF Databases
* z/TPFDF automatically organizes the database in a preferred sequence so the programmer need not worry about re-sequencing or sorting a Data file
* z/TPFDF also comes with the capability of automatically adjusting the file space based on the data in the files

**DISADVANTAGES**

* z/TPFDF uses a large amount of core and internally issues a number of z/TPF Macros to manage the data levels. Recent releases use DECB and hence these overheads are now minimal
* Enters a number of z/TPFDF Programs (Beginning with UFxx). Recently released CEP (Common Entry Point) feature of z/TPFDF ensures that these overheads are also reduced to a minimum
* Increase in Number of Instructions executed and Processing time

## z/TPFDF COMPONENTS

Basic Components of z/TPFDF are

* DBDEF Table, DBDEF Macros and DSECT Marcos
* DBIFB Block
* Macros and Functions
* Maintenance and Test Utility (ZUDFM)
* Data Collection Utility (ZUDFC)
* Capture/Restore utility, information and statistics environment (ZFCRU)
* Recoup Utility
  + **DBDEF TABLE, DBDEF MACROS AND DSECT MARCOS**
* DBDEF Table is generated using
  + DBDEF Macro instruction with the parameters that describes the file to z/TPFDF
  + DSECT Macro that describes the file and LRECs layout
* It provides the central definition for the database
* It holds information about the Location, Organization, characteristics of a file
* There is one DBDEF table and one Assembler DSECT macro for each file ID.
* DBDEF Macro contains information like File ID, name of the DSECT
* DBDEF Table is assembled output of DBDEF Macro.
* It is loaded on the online system,

* + - **DBIFB BLOCK**
* Whenever application opens a file using DBOPN or dfopn for first time, a work area called DBIFB (DataBase InterFace Block) is created
* DBIFB block contains SW00SR slot, which contains information about an individual sub-file.
* This SW00SR slot is created when DBOPN is issued.
* Next macro or function will move relevant information from DBDEF table into SW00SR slot.
* Fields of SW00SR slot are used by z/TPFDF to return information to the application program
* Address of DBIFB block is stored at **CE1DBS**.
* The SW00SR slot is returned when the z/TPFDF file is DBCLS'ed.
* The count of the number of z/TPFDF files open is maintained in the ECB field **CE1DBO**.
* A system error (SERRC DB0112) will occur at ECB EXITC time if any TPFDF files are open.
  + - **MACROS AND FUNCTIONS**
* Programmer code high-level z/TPFDF macros or functions to retrieve or data.
* These macros call online z/TPFDF, which uses the information in DBDEF tables to generate traditional z/TPF file handling requests.
  + - **MAINTENANCE AND TEST UTILITY (ZUDFM)**
* This provides maintenance and test capabilities that include:
  + Initializing files
  + Displaying and modifying files.
    - **DATA COLLECTION UTILITY (ZUDFC)**

This command is used to gather and display statistics related to system usage.

* + - **CAPTURE/RESTORE UTILITY, INFORMATION AND STATISTICS ENVIRONMENT (ZFCRU)**
* ZFCRU commands provide capture and restore capabilities that include:
  + Capturing files
  + Copying files
  + Logging files
  + Printing files
  + Restoring files
  + Validating files.
    - **RECOUP UTILITY (ZRECP)**

This facility is provided to recover usable long-term pool addresses, which are not returned to the system.

# TPFDF DEFINITIONS AND BASIC TERMS

This section will discuss about the basic definitions used in z/TPFDF

Following terms will be discussed in this section

* Basic Definition used in z/TPFDF
  + z/TPFDF File
  + Sub-File
  + Blocks
  + Logical Records.
  + z/TPFDF Header
  + z/TPFDF Trailers
* Type of LRECs
  + Fixed Sized LRECs
  + Variable Sized LRECs
* Algorithm
* Data Level Usage
* Defining z/TPFDF files and LREC's
* DBDEF

## 2.1 z/TPDFDF DEFINITIONS.

### z/TPFDF FILE:

* Physically a z/TPFDF File consists of several Sub-files having same record ID and sharing the same DSECT.

### SUB-FILE:

* A Sub-file Consists of a Prime Block (Fixed or Pool) and its Overflow Blocks.
* All z/TPF Files (Prime and Overflow) of a z/TPFDF Sub-file have the same Record ID and have similar layouts (DSECTS) and LRECs.
* Every z/ TPFDF Operation is performed at Sub-file level.

### BLOCKS

* Collection of logical records builds a block.
* When volume of data becomes more than that can be stored in the prime block, then z/TPFDF automatically allocate overflow block to hold extra data.
* All physical block contains same file-ID.
* Chaining of the blocks is internally managed by updating the Forward Chain of the Sub-file

### LOGICAL RECORDS (LRECs):

* Data in z/TPFDF files is organized in into logical groups called LRECs
* Smallest unit of data that an application program can read, write or add.
* LRECs cannot span blocks.

File ID

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | Primary Key or LREC ID |  |   BLOCK HEADER |
| |  | | --- | |  |   DATA Fields  LRECS |
| |  | | --- | |  |   DATA Fields |
| |  | | --- | |  |   DATA Fields |
| |  | | --- | |  |   DATA Fields |
| Block Trailers (Optional) |

**LRECs IN A BLOCK**

### STANDARD TPFDF HEADER

* z/TPFDF has 26 bytes of Header (16bytes Standard z/TPF+10bytes z/TPFDF extension).
* Out of these, first 16 byte is standard z/TPF header and contains following fields

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **LENGTH IN**  **BYTES** | **FUNCTION** |
| Record ID | 2 | 2-Byte Unique Identification Code for identifying the record |
| Record Code Check | 1 | Used for integration of the database |
| Control Byte | 1 | Tells about the record |
| Program Stamp | 4 | Stamp of program that filed the record last |
| Forward Chain | 4 | Address of Forward Chain |
| Backward Chain | 4 | Address of Backward Chain |

* Standard z/TPFDF header is 10 bytes long. It contains following fields
* Next Available Byte (NAB) – 2 Bytes (Maintained by z/TPFDF)
* Sequence Number for File updates – Internal Use – 2 Bytes
* TPFDF Work area – Internal Use – 6 Bytes

### z/TPFDF TRAILER

* z/TPFDF block also contains Optional z/TPFDF Trailer.
* This provides useful information for debugging programs.
* It is 36 bytes long and contains following fields.

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **LENGTH** | **PURPOSE** |
| Program Base Index | 2 | Helps in locating file address of programs |
| Database Index (DBI | 2 | Helps in locating subsystem FACE table to resolve file address of data record |
| SSI (Sub System ID) | 2 | Helps in locating unique set of fixed records (address of that subsystem Face table is find by DBI) |
| Program Name | 4 | Last program which filed the record |
| Spare | 4 | Free for further use |
| CPU ID | 1 |  |
| I-stream Number | 1 | On which I-stream it is executed |
| Time of Day Clock | 8 | Time when record was last filed |
| Original Block Check Character | 1 |  |
| Command Code for last issued command | 1 | Last command done for this file |
| File Address of Prime Block | 4 | File Address of Prime Block of the file |
| File Address of Current Block | 4 | File Address of Current Block |

## TYPES OF LRECs

* All the LRECs are identified with a Primary Key.
* Single z/TPFDF file can contains LRECs with different Primary Keys
* Each Primary key indicates different kind of data and has different fields
* LREC Primary Keys can take values from X’10’ to X’FE’.
* Primary Keys X’00’ to X’09’ and X’FF’ are reserved for the System

LREC are basically of 2 types

### FIXED-LENGTH LREC

* In this, all LRECs in a particular file is of same size
* It is not flexible and should be avoided
* In this 1st byte is used as primary key and is used to indicate the type of data stored in the LREC.
* Remaining field contains the LREC Data

### VARIABLE-LENGTH LREC

* In this, first 2 bytes indicate length of the LREC followed by 1byte identification Key (or Primary Key). Then comes the LREC data
* This type of LREC can contain only 1 variable Length field.
* Variable Length field should be last fields contained in an LREC, since it would have a calculable default length
* This is preferred type

## ALGORITHM

* It is an input data string used to calculate the correct ordinal number to access particular sub-file.
* This is used if Prime Block or Sub-file is fixed file.
* File type of the TPFDF file is specified in SW00RCT.
* TPFDF provides different algorithm, one of which should be chosen at the design stage and coded in DSECT in SW00RBV field
* It is chosen based on how many sub-files are required to distribute data within the TPFDF file and performance requirement for the database.
* TPFDF interface macros translates Algorithm into ordinal number based on
* SW00RBV,
* SW00BOR (beginning Ordinal number) and
* SW00EOR (Ending Ordinal number) defined for the File Type (SW00RCT)

**EXAMPLE of ALGORITHM:**

A TPFDF File of Departure Cities is designed such that

* Ordinals 0..25 will contain City Names, beginning A..Z respectively.
* There will be 26 Ordinals defined one for each Alphabet.
* 26 Sub-files will be created. One for Cities starting with each Alphabet.
* The first character of the name will form the Algorithm String passed to the command macros.
* The Algorithm code to translate the character ‘A..Z’ to its ordinal (0..25) is provided by TPFDF .

## DATA LEVEL USAGE

* The z/TPFDF product uses data event control blocks (DECBs) instead of data levels when processing most macros and functions.
* For z/TPFDF macros and functions that do not use DECBs, all data levels holding blocks are preserved.
* The z/TPFDF product does not alter the contents of data levels across z/TPFDF calls except as follows:
* If you use the DBDSP macro:
* Data level 1 (D1) and data level 3 (D3) are not data level independent 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 data level independent.
* TPFDF reserves Register R3 for its internal use as SW00SR base

## DEFINING TPFDF FILES AND LREC'S

* A TPFDF File is defined at two places
* The TPFDF File DSECT / Data Macro
* The Database Definition or the DBDEF
* The DSECT describes the logical record (LREC’s) and the user fields in LREC’s,
* Additionally the following information is also declared as part of the File DSECT
* Record ID of TPFDF File
* The Size of the Blocks (Prime, Overflow)
* The File record type
* Algorithm to be used

Given below is the extract of a TPFDF DSECT Definition

&SW00WID SETC 'C3' FILE ID &SW00BOR SETC '0' BASE ORDINAL NUMBER

&SW00EOR SETC '-1' END ORDINAL NBR

&SW00WRS SETC 'L1' BLOCK SIZE

&SW00ARS SETC 'L4' ALTERNATE BLOCK SIZE

&SW01EO# SETC '&SW00EOR' RECOUP END ORDINAL

&SW00RCT SETC '#EVACLASS' FACE FILE TYPE

&SW00RBV SETC '#TPFDB02' FILE ALGORITHM

&SW02FIL SETC 'DFC3BR' FILE DSECT NAME

## NAME CONVENTION FOR TPFDF FILE NAMES

* TPFDF File names have to follow a defined convention
* TPFDF File DSECT Names have to be exactly 6 Characters Long as depicted below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | A | N | N | C | C |

APPLICATION TYPE FILE IDENTIFIER COMPANY CODE

* The First Character identifies the Type of application the DSECT Belongs to.

Example

* + A Accounting
  + C PAX
  + F Fare quote/ticketing
  + G General functions
  + L Host to Host
  + M Message switching
  + R Passenger reservation
  + S System software
  + W Departure control system
* The Second Character identifies the type of File
  + G General Data Set
  + P Traditional (Non-TPFDF) File
  + R Real time Files (Prime Fixed or Pool)
  + W Work File , Exists Only for Life of an ECB
  + T Temporary LREC stored in W-type File
* The Third and Fourth Characters form a unique identifier that makes the FILE Unique across a group of TPFDF Files belonging to a particular application
* The Fifth and Sixth Characters form a 2- Character Company code from where the DSECT originated From. Usually at Airline Companies these would be the Airline Code.
* Example:
  + RR01BR, Is a TPFDF Real time file (Fixed or Pool) used by Reservations group at EVA Airlines
  + RWT2BR Is a TPFDF Work file used by Reservations used at SIA

## DBDEF

* The Database definition or the DBDEF is a macro coded in an E-type program by the Database administrator for each TPFDF File
* The DBDEF Programs are assembled and loaded to the zTPF System and forms a Central DBDEF Table that resides in core
* The Central DBDEF Table is looked up by the TPFDF Programs when applications perform operations on TPFDF Files
* The DBDEF Table is also used by other TPFDF Components such as the Online utilities, CRUISE, Data collection and Recoup
* The DBDEF picks up File related data such as record ID, Record type, Block sizes, Algorithm etc.. From the File DSECT.

**EXAMPLE**

DBDEF FILE=RRC1BR,

(ITK=#RRC1K80,ID2=(CHK0),

RID=RRC2BR,ADR=RRC1DLR-RRC1BID)

**EXAMPLE OF A TPFDF DSECT (RELEVANT FIELDS DISPLAYED)**

.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* .\* DEFINITIONS FOR TPFDB \*

.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

&SW00WID SETC 'C1' FILE ID

&SW00BOR SETC '0' BASE ORDINAL NUMBER

&SW00EOR SETC '-1' END ORDINAL NBR

&SW00WRS SETC 'L1' BLOCK SIZE

&SW00ARS SETC 'L1' ALTERNATE BLOCK SIZE

&SW01EO# SETC '&SW00EOR' RECOUP END ORDINAL

&SW00RCT SETC '#EVACLASS' FACE FILE TYPE

&SW00RBV SETC '#TPFDB02' FILE ALGORITHM

&SW02FIL SETC 'RRC1BR' FILE DSECT NAME

&SW00OP1 SETC '00000000' OPT BYTE1 (SEE SRGA-09-SAMP DOCUMENT)

&SW00OP2 SETC '00000110' OPT BYTE2 (SEE SRGA-09-SAMP DOCUMENT)

&SW00OP3 SETC '00000000' OPT BYTE3 (SEE SRGA-06-SAMP DOCUMENT)

&SW00TQK SETC '15' HIGHEST TLREC

.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* STANDARD ACPDB HEADER \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RRC1HDR&CG1 DS CL16 STANDARD FILE HEADER

DS CL10 STANDARD ACPDB HEADER

RRC1VAR&CG1 EQU \* START OF VARIABLE USER-AREA

RRC1HDL&CG1 EQU RRC1VAR&CG1-RRC1HDR&CG1 HEADER-LENGTH UP TO CR8IVAR

ORG RRC1HDR&CG1

RRC1REC&CG1 DS 0CL1 1ST RECORD START (1=VARIABLE,ELSE SIZE)

RRC1SIZ&CG1 DS H SIZE OF LOGICAL RECORD

RRC1KEY&CG1 DS X LOGICAL RECORD IDENTIFIER

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EQUATE OF LOGICAL RECORD KEYS (KEY AND LENGTH) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#RRC1K10 EQU X'10' LOGICAL RECORD KEY X'10'

#RRC1K80 EQU X'80' LOGICAL RECORD KEY X'80'

#RRC1L10 EQU RRC1E10&CG1-RRC1REC&CG1 LENGTH OF LOGICAL RECORD X'10'#RRC1L80 EQU RRC1E80&CG1-RRC1REC&CG1 LENGTH OF LOGICAL RECORD X'80'

RRC1ORG&CG1 EQU \* START OF LOGICAL RECORD DESCRIPTION

.\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEADER ITEM KEY = X'10' \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RRC1PAR&CG1 DS CL2 DATE OF LAST UPDATE (PARS DATE)

RRC1CRI&CG1 DS CL3 LAST UPDATE CRI

RRC1E10&CG1 EQU \* END OF LOGICAL RECORD WITH KEY = X'10'

ORG RRC1ORG&CG1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KEY = X'80' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RRC1NAM&CG1 DS CL11 PAX NAME

RRC1STA&CG1 DS X STATUS OF PAX

RRC1PCS&CG1 DS CL3 NUMBER OF PIECES

RRC1ORG&CG1 DS CL3 ORIGIN CITY

RRC1DES&CG1 DS CL3 DESTINATION CITY

RRC1DLR&CG1 DS XL4 PAX DETAIL RECORD FILE ADDRESS -> RRC2BR

RRC1E80&CG1 EQU \* END OF LOGICAL RECORD WITH KEY=X'80'

ORG RRC1ORG&CG1

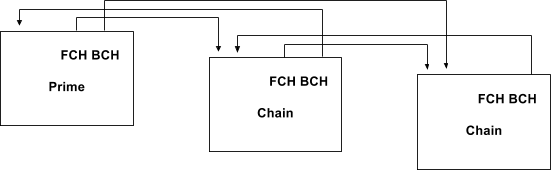
.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## ORGANISATION OF LRECs

* LRECs can be organized within a file in a particular order ascending or descending or without any order at all
* LRECs can be organized UP, DOWN or NOORG based on Particular key.
* KEY can be a Combination of up to 6 fields of an LREC (Up to 180 Keys if a keylist is used)
* If Organization is UP, LRECs are stored in ascending order of Key fields.
* If DOWN Organized, LRECs are stored in descending order of Key fields.
* NOORG Organization specifies that no particular order is maintained when LRECs are stored
* The Organization can be coded in the DBDEF or can be specified on TPFDF Command macros
* Default Key Combinations can be defined for ADD Operations in the DBDEF.
* Default key organization will supersede command macros key options for ADD of LRECS.
* KEY Organisations once set will be in effect till an application issues a command macro with a different Organisation or a NOKEY Option is used on a command macro that nullifies the currently active KEYS.

## TPFDF FILE HANDLING

* + 1. **CHAINING**



* Each FCH field contains the file address of the next chain record. Note that different block sizes for prime and chain can be used.
* FCH field in last chain is zero.
* Each BCH field contains the file address of the previous chain (or prime) record. This is an optional feature controlled by DSECT bit @SW00OP1 bit 0. There is extra I/O overhead required by z/TPFDF to maintain backward chains if this option is used.
* BCH field in prime contains the file address of the last chain. This is always present, irrespective of the above backward chain option.
* Logical records are not split between physical records.
  + 1. **DATA SHUFFLING**
* When logical records are DBDEL'ed, TPFDF will, "Shuffle" up the subsequent logical records in the same block.
* Data is only moved within the block, and creates empty space at the end of the block.
  + 1. **PACKING**
* Packing factor represents the maximum percentage of data that is allowed to be present in any block for that filetype, while the file is OPEN'ed.
* If a DBADD of a new logical record to a block causes chaining, the data in the file will be distributed among blocks according to the packing factor.
* Enough data from each block is moved to a new chain so that the amount of data in each block is below the packing factor.
* This means that more than one chain block could be created.
* If a DBDEL of a logical record(s) caused the available space in a block to go below the packing factor, TPFDF will compress the file at the DBCLS.
* This will move data between blocks and cause I/O.

## FIELDS OF SW00SR THAT PROGRAMMER SHOULD KNOW

|  |  |  |  |
| --- | --- | --- | --- |
| **SW00SR FIELD** | **DISPLACEMENT IN SW00SR** | **LENGTH** | **DESCRIPTION** |
| SW00WID | 010 | 2 Bytes | Record ID of required File |
| SW00FAD | 044 | 4 Bytes | File address of Prime Record – 4 |
| SW00RTN | 081 | 1 byte | TPFDF Command Return Indicator |
| SW00REC | 084 | 4 Bytes | Address of Located Record |
| SW00WKA | 260 | 4 Bytes | Address of Workarea |

# TPFDF FILE COMMANDS AND COMMAND SEQUENCE

This section will discuss about the file commands and commands sequence

Following terms will be discussed in this section

* Functions
* z/TPFDF File
* Sub-File
* Blocks
* Logical Records.
* z/TPFDF Header
* z/TPFDF Trailers
* Type of LRECs
* Fixed Sized LRECs
* Variable Sized LRECs
* Algorithm
* Data Level Usage
* Defining z/TPFDF files and LREC's
* DBDEF

## FILE COMMANDS

* The Following operations can be performed on a TPFDF File.
* These commands normally work at a Sub file Level, and optionally on a range of Sub files or the entire TPFDF File

|  |  |
| --- | --- |
| **FUNCTION NAME** | **PURPOSE** |
| FILE OPEN | * This Command is the Basic Command * It is a Pre-requisite to most of the TPFDF Command Macros, * Sets up the SW00SR slot for the File |
| READ | To Read an LREC From an opened Sub file |
| ADD | To Add an LREC to an opened Sub file |
| DELETE | To Delete an LREC(s) from a Sub-file |
| MODIFY | To Modify an LREC that has been read previously |
| REPLACE | To replace an LREC which is read previously with a new LREC |
| DISPLAY | To Display LRECs From the current Sub-file |
| CLOSE | * To stop processing a Sub-file, * Clears SW00SR Slot |
| CREATE | * Get a Pool Prime Record and * Create a New TPFDF Sub-file. * Equivalent of a GETFC Macro. |
| CLEAR | * Exit without CLOSING opened sub-files. * LREC Modifications not written to DASD and are lost |
| RETAIN | * To save File address, Core address and Displacement in the block of the current LREC |
| SPACE | Allocate free workarea space in the DBIFB For an opened Sub-file |
| DBIFB | * To Find base address of any SW00SR Slot which has been opened previously * Used to check if particular file is opened or not |
| DBADR | * To calculate ordinal number given an Algorithm argument. * Used to select a Range of Sub-files for TPFDF Operations |
| COPY | To Copy a Sub-file into Pool blocks |
| RESTORE | Restore a Sub-file previously copied using TPFDF COPY Command |
| CHECKPOINT | * To save Blocks of opened Sub-file into DASD * Issues FILNC Macro’s on all blocks of the opened Sub-file |
| SORT | Sort LRECS From an input Sub-file onto another Sub-file |
| MERGE | To Merge LRECS From input Sub-file into a Target Sub-file |
| FREE LEVEL | Free a data level from TPFDF For normal TPF/zTPF Use |
| DBKEY | To define a list of keys for use with subsequent TPFDF Command macros with a Sub-file |
| UNIQUE KEY | To get a Unique 4-byte Key value to use in LRECS |

## COMMANDS SEQUENCE

* OPEN Should be the First Command for a sub-file
* MODIFY, REPLACE Should follow either a READ or an ADD, i.e. Current LREC pointer should have a non-zero value
* DBRET must be done only after DBRED has been done
* All opened sub-files should be Closed before EXIT
* Invalid Command Sequences can cause TPFDF dumps or unexpected results

# BASIC z/TPFDF CODING CONCEPT

This section will discuss about the basic concept that to be taken into care while coding the programs using z/TPFDF macros

Following terms will be discussed in this section

* Coding a TPFDF Command Macro

It provides basic details about parameters and basic concepts of the programming.

* Common Parameters Used

Details about parameters used for describing the file to be accessed.

* Parameters to pick one sub-file

Provide parameters to calculate address for accessing the particular sub-file

* Parameters to Specify LRECs to be operate

Basic details about the

* Defining z
* DBDEF



## CODING A TPFDF COMMAND MACRO

* A TPFDF Command macro consists of the
  + Command macro and
  + its Optional and
  + Mandatory Parameters
* Parameters are either Positional or Keyword
* Most of the Parameters are commonly used across many of the TPFDF Command macros
* TPFDF command macros whose parameters exceed one line in the Program should be continued in the next line.
* A continuation character should be entered in Column 72
* The continued line's last character should be a comma,
* The continued line should begin at column 16
* Command Macros can be coded within SPM Constructs such as DO Loops and IF-THEN-ELSE Loops

## COMMON PARAMETERS USED

**"REF =" PARAMETER:**

* It is used to specify the file that you want to access
* Code a DSECT name of the TPFDF File or a reference to the DSECT name,
* Either an 8-byte field containing the file reference name or a 4 byte field containing the address of the reference name
* When coded alongside the TPFDF OPEN Command,
  + Allocates a SW00SR Slot for the opened Sub-file and
  + also loads register R3 With the SW00SR Slot address,
* The same reference is used to refer to the Sub-file for subsequent commands on the Sub-file
* The DSECT Name makes up the Sub-file reference for the SW00SR Slot in the SW00SR Block

**FILE =" PARAMETER**

* If the First 6 Characters of the REF= Parameter is not the name of the DSECT then it is mandatory to code the FILE= parameter and supply the DSECT Name through it
* Using FILE= on command macros means that the application has set up R3 to point to the required SW00SR slot on its own behest.
* It is advisable not to use this parameter, instead use the REF parameter to specify the file you want to access.

"**SUFFIX =" PARAMETER:**

* Code a character as an argument,
* Generates using statement for the DSECT With the suffixed character

## PARAMETERS TO PICK ONE SUB-FILE

|  |  |  |
| --- | --- | --- |
| **PARAMETER NAME** | **PURPOSE** | **EXAMPLE** |
| ALG= | * Code an Algorithm string value here that pertains to a Sub file, * It can be a Label, an A/Label, Register or an Absolute value can be specified here. | * DBOPN REF=RRC1BR,ALG==C'A' * DBOPN REF=RRC1BR,ALG=EBW000 * DBRED REF=RRC1BR, REG=R7,ALG=EBW000 |
| ORD= | * It contains 4-byte field containing Ordinal number of a fixed file * It can be label or A/label | * DBOPN REF=RRC1BR,ORD=EBX024 * DBOPN REF=WR99BR,REG=R5,   ORD=INPORD |
| FADDR= | * It contains label of 4-byte location that contains file address of Sub-file | DBOPN REF=RRC2BR, FADDR=RRC1DLR  file addr is in the field RRC1DLR |

## PARAMETERS USED TO SPECIFY LREC (S) TO OPERATE

* **DEFINITIONS**

A **search key** is a field, or consecutive fields, within logical records, which are used as an argument in a compare or "test under mask" conditional test against an application supplied search argument.

The first byte of a variable size TPFDF logical record is called the ***primary key***.

* **"MATCHING"**
  + Keys are used to match on data fields when searching through the logical records to read or update the database.
  + The primary key should always be the first search argument.
  + Any field or consecutive fields within a logical record can be used as a search key.
* **ORGANIZATION**
  + Primary key is used to distinguish between different types of logical records within the same file.
  + Within LRECs of particular key, to maintain ease with which data is searched, usually LRECs are arranged according to other fields. This arrangement is called Organization

LRECs are arranged according to one of following 3 organizations

* + **UP ORGANIZED.**

In this, particular sub-field will be arranged in ascending order.

For Example

|  |  |  |  |
| --- | --- | --- | --- |
| SIZE | KEY | CARRIER NAME | FLIGHT NUMBER |
| 10 | 80 | AI | 0150 |
| 10 | 80 | AI | 0200 |
| 10 | 80 | BR | 0300 |
| 10 | 80 | BR | 0400 |

In this case, LRECs are UP Organized based on CARRIER NAME and FLIGHT NAME.

Or in Other words, CARRIER NAME and FLIGHT NUMBER are UP Organized.

* + **DOWN ORGANIZED**

In this, particular sub-field is arranged in descending order

|  |  |  |
| --- | --- | --- |
| SIZE | KEY | RATING |
| 10 | 80 | 10 |
| 10 | 80 | 8 |
| 10 | 80 | 6 |

In this case, LRECs are DOWN organized based on RATING

* + **NOORG (or NO ORGanisation)**

In this there is no order in which key is added

|  |  |  |
| --- | --- | --- |
| SIZE | KEY | CARRIER CODE |
| 10 | 80 | AF |
| 10 | 80 | KL |
| 10 | 80 | BA |

Default Organization is NOORG

## BASIC POINTS RELATED TO THE KEY PARAMETERS

* Upto 6 Key Parameters KEY1, KEY2..KEY6 can be set up with a TPFDF Command macro,
* These must be specified in ascending order and are connected with Boolean AND Operation
* If KEYn Parameters are specified Organization of the keys also should be specified,
* Once Keys and Organization are setup for a Sub file it remains in effect until
  + another TPFDF macro is issued with another set of Key parameters or
  + Keys are nullified using the NOKEY Parameter

## SUB-PARAMETER USED WITH KEYn PARAMETER

|  |  |  |
| --- | --- | --- |
| **SUB-PARAMETER NAME** | **PURPOSE** | **EXAMPLE** |
| **R=** | * Specified Field with in LREC to be compared with Search argument specified with S= * It can be   + Label   + D/Displacement   + D/decimal displacement | * DBRED   REF=RRC1BR, UP,  KEY1=(R=RRC1KEY, S=X'80'),  KEY2=(R=RRC1NAM, S=MI0ACC+10)   * DBRED   REF=RRC1BR, UP,  KEY1=(R=RRC1KEY, S=X'80'),  KEY2=(R=D/RRC1NAM-RRC1KEY,S=MI0ACC+10)   * DBRED REF=RRC1BR,UP,   KEY1=(R=RRC1KEY,S=X'80'),  KEY2=(R=D/3,S=MI0ACC+10) |
| **S=** | * Specified the value to compare with against the field in the LREC specified with “R- Subparameter * It can be * Label, * A/label, * register (containing address of search argument), * equate, * I/Value or a Literal. | * DBRED REF=RRC1BR,UP,   KEY1=(R=RRC1KEY,S=#RRC1K80),  KEY2=(R=RRC1NAM,S=MI0ACC+10)   * DBRED REF=RRC1BR,UP,   KEY1=(R=RRC1KEY,S=X'80'), KEY2=(R=RRC1NAM,S==C'AB123456789') |
| **L=** | * Length of data to compare. * Default is length of target field * It can be   + label of 2-byte field   + L’labels   + Halfword literal | DBRED REF=FRS1BR,UP, KEY1=(R=FRS1KEY,S=#FRS1K80),  KEY2=(R=FRS1ALC,S=MI0ALC), KEY3=(R=FRS1CNA,S=MI0CNA,L=EBW000), KEY4=(R=FRS1FLN,S=MI0FLN,L=L'WRK0ALC+L'WRK0FLN),  KEY5=(R=FRS1BPT,S=MI0BPT,L==H'3') |
| **M=** | * Specifies 1-byte mask as search argument * It should be immediate value | DBRED REF=RRC1BR,  KEY1=(R=RRC1KEY,S=#RRC1K80,UP), KEY2=(R=RRC1NAM,S=MI0ACC+10,UP),  KEY3=(R=RRC1STA,M=X'20',C=O,NOORG) |
| **D=** | * It specify 1-byte Dynamic mask * It is exclusive to M= * It can be label of address of 1-byte field | DBRED REF=RRC1BR,  KEY1=(R=RRC1KEY,S=#RRC1K80,UP), KEY2=(R=RRC1NAM,S=MI0ACC+10,UP),  KEY3=(R=RRC1STA,D=EBX033,C=O,NOORG) |
| **C=** | Specified condition to be satisfied.   * Compare code with S=   GT.LT,NE, EQ, NH, NL, GE, LE, E, H, L (Default = EQ)   * Compare code with D= or M=   Z, O, M , NZ, NO, NM (it is mandatory) | DBRED REF=CR7MKL,REG=RGD, 02>  KEY1=(PKY=#CR7MK80,UP), KEY2=(R=CR7MTYP,S=CI7MATS,C=NE,NOORG) |
| UP|DOWN|NOORG | It specifies organization of Key   * DOWN means key field is in Descending order in Sub file * UP means Ascending Order * NOORG means no order and it is default * Do not use UP or DOWN after NOORG * Always watch for these statements if you have not followed KEY Organisation rules, MNOTES will be generated to Warn/Inform you * A General Organisation Common to all Keys can be specified as a main parameter with the Command Macro * Default Key Organisation for D=/M= Keys and all Subsequent Keys Should be NOORG | DBRED REF=RRC1BR  KEY1=(R=RRC1KEY,S=#RRC1K80,UP), KEY,NAM **UP organised** KEY2=(R=RRC1NAM,S=MI0ACC+10,UP), in ascending order KEY3=(R=RRC1STA,D=X'FF',C=O,NOORG) Status not orgnised  DBRED REF=RRC1BR,  KEY1=(R=RRC1KEY,S=#RRC1K80,UP), KEY,NAM UP organised KEY2=(R=RRC1NAM,S=MI0ACC+10,UP), in ascending order  KEY3=(R=RRC1STA,D=X'FF',C=O,NOORG), status not organised  KEY4=(R=RRC1ORG,S==C'MAA',C=NE,NOORG) origin not organisd |
| NOKEY | * Deactivate any previously active keys * Process without any KEYS all LRECs are processed starting from current LREC | READ NEXT CLX RECORD 05\*  DBRED REF=CR34KL, NOKEY, REG=RGF |
| ERROR= | Specify SPM label where control should branch if there is any serious error in executing command | DBRED REF=CR7AKL, UP, REG=RGD, X  KEY1=(R=CR7ABRD,S=CI7DCIPA), X  ERROR=TPFDFERR  #LOCA TPFDFERR |
| ERRORA= | Specify Assembler label where control should branch if there is any serious error in executing command | DBIFB REF=JR1UKL,ERRORA=OFH5ABO2  DBCLS REF=JR1UKL,ABORT,RELEASE  OFH5ABO2 EQU \* ELSE |
| REG= | Specifies base address of LREC that has been processed | DBRED REF=CR34KL,NOKEY,REG=RGF |
| PKY | Specifies the Primary Key | DBRED REF=CR7MKL,REG=RGD, 02>  KEY1=(PKY=#CR7MK80,UP), KEY2=(R=CR7MTYP,S=CI7MATS,C=NE,NOORG) |

## CHECKING RETURN CONDITIONS OF TPFDF COMMAND MACROS

The Following Return codes can be checked on return from TPFDF Command macros

**SW00RTN CONDITIONS RETURNED FROM TPFDF COMMANDS**

|  |  |
| --- | --- |
| **SW00RTN RETURN CODE** | **CONDITION ENCOUNTERED** |
| #BIT0 ON | I/O Error |
| #BIT1 ON | LREC Not Found |
| #BIT2 ON | FACE return error |
| #BIT3 ON | ALGORITHM error |
| #BIT4 ON | Data in Block is Corrupted |
| #BIT5 ON | EOF During FullFile Processing |
| #BIT6 ON | Sequence error using DBRST macro |
| #BIT7 ON | Sort/Merge error |

TPFDF has defined equates to test SW00RTN Values

|  |  |
| --- | --- |
| **RETURN VALUE SW00RTN** | **CONDITION** |
| #TPFDBEX (X'AB') | Any Serious error except indexing errors |
| #TPFDBER (X'BB') | Any Serious error |
| #TPFDBNR (X'44') | LREC not found |
| #TPFDBOK (X'00') | No error , Record found |

TPFDF Provides short form of testing SW00RTN using SPM's as follows

|  |  |
| --- | --- |
| DBFOUND(YES|NO) | Check if LREC Found |
| DBERROR(YES|NO) | Check if error occurred |
| DBEOF(YES|NO) | Check End of file condition |
| DBIDX(YES|NO) | Check for INDEX errors |

**EXAMPLE**

#IF DBFOUND,YES

Code to process that LREC

#ELSE

DBCLS REF=WR5RCO

#EIF

* SW00RT1 has the count of total errors while FULLFILE Processing
* SW00RT2 #BIT0 on means error in LIST= specification

#BIT1 FMSG/OMSG error during DBDSP

#BIT4 B+Tree Index Error

**NOTE** : Do not check return codes after a DBCLS Command Macro.

# TPFDF MACROS:

## DBOPN - OPEN A SUB-FILE

* The DBOPN is coded for a Sub-file before performing other operations on it,
* Creates SW00SR Slot in the DBIFB,
* Refers to the DBDEF,
* Also generates USING statement with the specified register and
* Generates SUFFIXED labels if SUFFIX Parameter is coded

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER** | **PURPOSE** |
| REF=, FILE=  (Mandatory) | Mandatory Parameter, as discussed in Section 4.2 |
| REG= | * Generates the DSECT macro specified with the REF or FILE parameter and generates a USING statement to provide addressability to the DSECT * If it is not mentioned, then programmer has to code it explicitly before using it. |
| SUFFIX= | Allow you to use same DSECT to map 2 different area of memory. |
| ALG=  ORD=  FADDR= | Optional Parameter described in Section 4.3 |
| BEGORD=  ENDORD= | Overrides Beginning and End Ordinal number to be used in subsequent DBRED macro. |
| WRAPAROUND | * It sets WRAPAROUND bit. * In this, if FULLFILE is used for any subsequent command, then sub-file with Ordinal number =0 will be processed after last ordinal number |
| HOLD /  NOHOLD | * HOLD parameter must be specified if the file is to be updated by the application. * If modification is done on a sub-file, which is not opened by HOLD, SERRC DB010C is issued. * This is also used when some file must be accessed by only one program like Queue records.   NOHOLD does not hold the sub file that you are accessing. |
| DETAC | * Each block of the Sub file read is kept in Main storage. * Updates are not written to Disk unless check-pointed or closed. * Used for faster access of LRECs, * Code for time critical applications. * Closing this Sub file with ABORT Parameter does not commit the changes made to DISK, * This is a unique commit mechanism provided by TPFDF. * Very Core Intensive parameter, use prudently |
| SPACE | SPACEB | * Used for requesting workspace that can be used as scratch area or to build records to be added. * You can specify the number of bytes (Immediate value) & Register to hold base address of work area, * If not specified the Application has to load the work area address from SW00WKA * SPACE initializes storage to X’00’ and * SPACEB initialized to X’40’ |

**EXAMPLES**

* **Specifying File name and base register**

DBOPN REF=WR5RCO, REG=R1

* **Specifying the access method**

DBOPN REF=WR5JCO, REG=R1, FADDR=EBW000 (Pool File)

DBOPN REF=WR5RCO, REG=R1, ALG=EBX000 (Fixed File)

DBOPN REF=WR5RCO, REG=R1, ORD=WA0ORD

* **Opening a file when update is intended.**

DBOPN REF=WR5RCO, REG=R1, ALG=EBX000, HOLD

(HOLD is MUST if file is to updated)

DBOPN REF=WR5RCO, REG=R1, ALG=EBX000, HOLD, DETAC

(in this all data blocks are detached and remain in core.

No blocks are filed until DBCLS or DBCKP is executed)

* **Getting Workarea space**

DBOPN REF=WR5RCO, REG=R1, ALG=EBX000, HOLD, SPACE=(340)

DBOPN REF=WR5RCO, REG=R1, ALG=EBX000, HOLD, \*

SPACE=(L'FR31DAT,R5)

DBOPN REF=WR5RCO, REG=R1, ALG=EBX000, HOLD, SPACEB=(80)

**NOTE:**

* Address of the SW00SR slot is returned in R3.
* DBOPN cannot be used with T-type DSECT as T-Type DSECT represents temporary logical record stored in W-type file.
* To open 2 or more instance of selected sub file of a particular file simultaneously, use suffix parameters or use different Reference name with same initial 6 characters.
* 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
* The z/TPFDF product uses data event control blocks (DECBs) instead of data levels when processing most macros and functions.
* For z/TPFDF macros and functions that do not use DECBs, all data levels holding blocks are preserved.

## dfopn–OPEN A SUB-FILE

**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);

|  |  |
| --- | --- |
| ref\_name | * Address of variable containing 8-byte reference name of the sub-file to be opened. * If less than 8 bytes are provided, it must be null-terminated. * z/TPFDF product will pad it with the blanks to build an 8-byte reference name |
| id | * Pointer to 2-byte identifier. * It can be either in character form or hexadecimal form |
| options | * DFOPN\_DETAC   + Opens file in DETAC Mode.   + All modified block are saved in the main storage   + No modified blocks are written to DASD until you checkpoint the sub-file (dfckp) or close the sub-file (dfcls) * DFOPN\_NODET   + Specifies that we don’t want sub-file to be open in detac mode * DFOPN\_HOLD   + Holds the sub-file to prevent 2 or more application program from modifying the sub-file at the same time   + Mandatory parameter if we are planning to modify the sub-file * DFOPN\_NOHOLD   + File can be accessed by more than 1 program.   + File is opened without any hold. * DFOPN\_NOCHK (In this RCC is not checked) |
| access | It is the method used to access the sub-file. It can be either of following   * DFOPN\_ALG * DFOPN\_FADDR * DFOPN\_FADDR8 * DFOPN\_ORD |
| acc | * It can be either algorithm, file address, ordinal number and its type depends on the value specified in access parameter |

**RETURN CONDITIONS**

* It returns the address of the SW00SR slot.
* Error condition in SW00RTN field of the SW00SR slot has no meaning for this function.

**EXAMPLE**

* The following example opens a sub-file that has DSECT name CW90KL.

The z/TPFDF product sets a pointer to the SW00SR slot in file\_ptr.

The file ID is "90".

dft\_fil \*file\_ptr;

file\_ptr = dfopn("CW90KL ", "90", DFOPN\_HOLD | DFOPN\_DETAC);

## DBRED - READ AN LREC

* The DBRED command reads an LREC from an opened Sub-file,
* The pointer to the LREC is loaded in the register specified using the REG= parameter,
* This is also known as the current LREC.
* If LREC is not found then DBRED sets a LREC NOT FOUND indicator in SW00RTN byte.
* After a Successful read, The LREC Fields in the LREC can be accessed using DSECT Labels,
* Subsequent DBRED will start reading from Current LREC downwards.

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER TO DESCRIBE THE FILE TO BE ACCESSED** | |
| FILE=  REF= | * They are same as used in DBOPN parameter. * There is no need of providing ID |
| **PARAMATERS TO ACCESS PARTICULAR SUB-FILE** | |
| ALG=  ORD=  FADDR= | * Same as used in DBOPN parameter * Refer section 4.3 for more details |
| **SEARCH PARAMETER** | |
| KEYn | * Please refer Section 4.6 for details about this parameter |
| **PARAMETER DECIDE FROM WHERE TO READ AND WHICH LREC TO READ** | |
| LRECNBR | * Describe the sequence number of LREC that we want to access |
| BEGIN | * Start searching from starting |
| BACKWORD | * Read LREC immediately preceding current LREC * SW00OP1, Bit0 should be set to indicate fill backward chaining |
| LAST | * Reads Last matching LREC in the sub-file |
| PREVIOUS | * Read the LREC that has been saved using DBRET * DBRET is (described in Section 5.) |
| * REG= is used to mention the register in which address of the matching LREC is stored | |
| **ERROR PROCESSING PARAMETERS** | |
| ERROR | * If there is a serious error, then processing branches to this SPM Label. |
| ERRORA | * If there is a serious error, then processing branches to this Assembler Label. |
| **FAST** is used to increase processing speed of the DBRED   * It is usually used for migration, where fast processing is required * In this case, TPFDF generates inline code. * It deactivates previously active keys. Similar to NOKEY * Used when we are intending to read all LRECs in Sub-file * Cannot be used with KEYn and LRECNBR parameters | |
| **PARAMETER TO BE USED WITH FULLFILE** | |
| BEGORD | It is starting ordinal number of the sub-file to be searched |
| ENDORD | It is ending ordinal number of the range of sub-file to be searched |
| HEADER | It locates header of next sub-file |
| **PARAMETER USED FOR READING LREC SAVED USING DBRET** | |
| STACK | Location of 10-byte field that contains the details about LREC |
| STACKREF | Value assigned to retained LREC |

**EXAMPLES:-**

* TO READ NEXT RECORD IN THE SUB-FILE

DBRED REF=RRC1BR, REG=R4 (No KEY parameter is mentioned)

* TO READ FIRST KEY80 ITEMS

DBRED REF=IWDWDF,REG=R4,BEGIN, (To indicate search from begin)

ERROR=ERR-IWD,

KEY1=(PKY=#IWDWK80) (indicates which key to be found)

NOTE: CODING PKY=#IWDWK80 is EQUIVALENT TO CODING

KEY1=(R=IWDWKEY,S=#IWDWK80,EQ)

PKY is recommended

* READ LAST MATCHING LREC

DBRED REF=QRHABR, REG=R5, LAST

* TO READ ALL ITEMS IN THE RANGE OF THE SUB-FILE

DBRED REF=RRPKBR,REG=R5,FULLFILE

* READ RECORDS THAT MATCH THE KEYS

DBRED REF=RR98BR,REG=R5,UP

KEY1=(PKY=#RR98K80),

KEY2=(R=RR98SLJ,S=RR98SLJD)

**NOTES:=**

* SW00RTN is used to contain error returned from DBRED.
* Address of current LREC is stored in SW00REC field and register specified in REG= also contains this value.
* When STACK, PREVIOUS parameter are used, sub-file must be opened with DETAC or HOLD (or both) option, else SERRC **DB017B** will be issued
* If no search parameters are set, DBRED locates next LREC in sequence in sub-file.
* If no STACK or STACKREF parameter is specified, details of only current LREC are retained.
* To ensure that an LREC is retrieved correctly when you use the [DBRED](http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.8/com.ibm.ztpf-ztpfdf.doc_put.08/bdfp1/mac_dbred.html?lang=en-us&cp=SSB23S_1.1.0.8) or [DBRET](http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.8/com.ibm.ztpf-ztpfdf.doc_put.08/bdfp1/mac_dbret.html?lang=en-us&cp=SSB23S_1.1.0.8) macro, adhere to the following rules when using retained LRECs:
* If you specify the STACK parameter or if you do not specify a retained LREC parameter (STACK or STACKREF),
* You cannot specify the STACKREF parameter on any subsequent [DBRED](http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.8/com.ibm.ztpf-ztpfdf.doc_put.08/bdfp1/mac_dbred.html?lang=en-us&cp=SSB23S_1.1.0.8) or [DBRET](http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.8/com.ibm.ztpf-ztpfdf.doc_put.08/bdfp1/mac_dbret.html?lang=en-us&cp=SSB23S_1.1.0.8) macro call to the same open file.
* If you want to use the STACKREF parameter on a subsequent macro call, you must close and reopen the file first.
* If you specify the STACKREF parameter,
  + - You **must** specify this parameter (not STACK) on all subsequent macro calls to the same open file.
    - If you do not want to specify a retained LREC parameter or if you want to specify the STACK parameter, you must close and reopen the file.

## dfred–READ ONE OR MORE LOGICAL RECORDS

* This is used to read a LREC,
* Read block header
* Get address where record is stored
* Read next LREC in sequence

**FORMAT AND PARAMETERS**

* 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*);

**(FOR DATA TYPE REFER FORMAT PROVIDED)**

|  |  |
| --- | --- |
| **FILE DESCRIBING PARAMETERS** | |
| file | * Pointer to base address of SW00SR slot of the file that we want to access * It is returned by dbifb or dfopn function |
| **SUB-FILE ACCESS PARAMETERS** | |
| access | * This describes the method that will be used to access sub-file * It can be one of the following   + DFRED\_ALG= acc parameter contains pointer to the algotrithm argument   + DFRED\_ORD = acc parameter contains ordinal number   + DFRED\_FADDR = acc parameter contains 4-byte FA |
| acc | * This describes the value used in conjunction with the ***access*** parameter |
| **PROCESSING OPTIONS** | |
| options | * DFRED\_FULLFILE= means   + Search in any of the sub-file in the file.   + Default is current sub-file,   + This parameter must be provided if you would like to specify search in any of the sub-file * DFRED\_BEGIN =   + Start searching from the start of sub-file * DFRED\_BACKWARD   + Read LREC just before current LREC position.   + Can’t be used with DFRED\_LRECNBR or when keys are active * DFRED\_LAST =   + Reads last LREC or last matching LREC * DFRED\_PREVIOUS=   + Reads LREC last retained by DFRET * DFRED\_HEADER   + Locates the sub-file header in the prime block   + SW00REC contains header address rather than LREC address * DFRED\_NOKEY   + Deactivates any currently active keys * DFRED\_INDEX\_HOLD   + Holds any index file that references sub-file that we are accessing   + Prevents 2 or more application from modifying index files at a same file   + Holding occurs if bit 4 and 5 in &SW00OP2 is set in the DSECT macro * 0   + Used when we don’t want to use any processing options |
| **PARAMETER RELATED TO LREC SEQUENCE NUMBER** | |
| * nbr\_type | * Provides the type of parameter provided in ***nbr*** parameter * It can have one of the following value * DFRED\_LRECNBR   + In this, sequence number of LREC to be read is provided in ***nbr*** parameters. * DFRED\_STACKREF   + Stack reference number is provided in nbr parameters. * DFRED\_STACK   + nbr parameter contains ptr. to 10-byte stack area |

**RETURN CONDITIONS**

* A pointer to the standard header of the block is returned If DFRED\_HEADER is specified
* A pointer to the LREC that is read is returned.
* SW00RTN=0 if specified LREC is located successfully.

**EXAMPLES**

* Simplest Way of reading dfred Next LREC

lrec\_ptr = dfred (File\_ptr,0);

Where Lrec\_ptr is pointer to the DSECT structure.

File\_ptr is pointer to SW00SR slot.

## DBCLS: CLOSE A SUB-FILE(S)

* The DBCLS command is usually the last command macro issued on a Sub file,
* Files all updates made to the Sub file to DASD,
* Need to code regardless of modification to Sub file to release blocks and SW00SR slot,
* All opened Sub files should be closed before EXITC,
* Can close more than One Sub file with one DBCLS

**PARAMETERS FOR DBCLS**

|  |  |
| --- | --- |
| **PARAMETER DESCRIBING FILE TO BE CLOSED** | |
| *REF= | FILE= (or both),* | * Same as discussed earlier. * REF=ALL can be coded to close all open Sub files. |
| **WHETHER TO COMMIT OR ABORT THE CHANGE** | |
| ABORT | * Closes sub file without filing blocks to DASD. * Can be used only when opened in the DETAC mode * All the updates since file was opened OR last DBCKP macro are discarded. |
| **WHAT TO DO WITH THE SW00SR SLOT** | |
| RELEASE  NORELEASE  REUSE | * RELEASE releases the SW00SR slot * NORELEASE   + Doesn’t releases the SW00SR slot and   + It will be used only if we are intended to process it sub file at later time. * REUSE   + Causes SW00SR slot to re-initialize so that it can be REUSED by same file and so DBOPN are not required.   + If SPACE parameter was used, then area will remain allocated. This option can be used to switch to other algorithm for the same file |
| **PARAMETER FOR RELEASING SUB-FILE** | |
| RELFC | * Release the Sub file and is equivalent to delete LRECs in that sub file * All File address and chain are released if prime is pool file. |
| **PARAMETER TO SPECIFY NEW REFERENCE** | |
| NEWREF | * Specify a new REF name and * It can be used with either NORELEASE or REUSE to rename the SW00SR slot reference. |
| **WHETER TO PACK OR UNPACK WHILE CLOSING FILE** | |
| PACK NOPACK | * If amount of LREC space left empty due to deleting LREC falls below a specified percentage in block, TPFDF packs the sub file if amount of LREC space falls below specified percentage * PACK forces that sub-file to be packed irrespective of available free space. * NOPACK restricts PACK operation |
| **INDICATE WHETHER FILES PROVIDED IN LIST NEEDS TO BE CLOSED OR NOT** | |
| EXCLUDE  INCLUDE | * These are used with REF=ALL parameter * EXCLUDE means close all **excluding** those mentioned in the list * INCLUDE means only files mentioned in the list are closed. IF any of the files specified is not OPENED, they are ignored. |
| **PROVIDING LIST OF FILE TO BE CLOSED OR NOT TO BE CLOSED** | |
| LIST | * USED with INCLUDE or EXCLUDE parameter. Provides the list of the files to be closed. * In this, location specified contains Number of the file to be closed. (Halfword) Followed by list of 8 characters REF name. |

**EXAMPLES:**

* **CLOSE ONLY RRC1BR AND WRX8BR FILES.**

DBCLS REF=ALL, INCLUDE=(RRC1BR, WRX8BR)

* **CLOSE ALL FILES EXCLUDING RRC1BR AND WRX8BR**

DBCLS REF=ALL, EXCLUDE=(RRC1BR, WRX8BR)

* **CLOSE LIST OF THE FILES**.

DBCLS REF=ALL, INCLUDE=(LIST=EBW000)

EBW000 (HALF-WORD) contains number of files in the list

After this list of 8-character ref name for each file is provided

* **PROVIDES LIST OF THE FILES THAT SHOULDN’T BE CLOSED**

DBCLS REF=ALL, EXCLUDE=(LIST=EBW000)

* CLOSE THE FILE AND RELEASE THE SW00SR BLOCK

DBCLS REF=WR5RCO, **RELEASE**

* Close the file, but don’t release and re-initialsed so that it can be used by same file and so dbopn is not released

DBCLS REF=WR5RCO, REUSE

* THE RELFC PARAMETER CAUSES ALL LOGICAL RECORDS TO BE DELETED AND THE FILE IS RELEASED.

DBCLS REF=WR5RCO, RELFC, RELEASE

**NOTES:**

* Never check error condition after DBCLS.
* Either SW00SR block is released OR even if SW00SR block is retained, then error indicators are not valid.
* If you specify RELFC parameters on DBCLS, DBDIX macro is issued if file is indexed.
* If no INCLUDE and REF=ALL is specified, Attempt to close file which is not opened, results in DB115 system error

## dfcls–CLOSE A SUB FILE

**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*);

**PARAMETERS**

**(FOR DATA TYPE REFER TO FORMAT GIVEN ABOVE)**

|  |  |
| --- | --- |
| **PARAMETERS TO DECIDE THE SUB-FILE TO BE ACCESSED** | |
| file | * Pointer to base address of SW00SR slot of the sub-file that we want to access * use NULL while using DFCLS\_ALL parameter |
| alg | Pointer to algorithm parameter to identity sub-file |
| **PARAMETER TO SPECIFY WHETHER TO INCLUDE OR EXCLUDE LIST OF SUB-FILE (list\_type)** | |
| DFCLS\_INCLUDE | Indicate that files mentioned in***lst*** contains files which should be included while closing |
| DFCLS\_EXCLUDE | Files mentioned in the ***lst*** contains the list of the file which should be excluded from closing |
| lst | Pointer to the structure that contains a list of files (to be closed or excluded from closing) |
| *option* ***PARAMETER TO DECIDE ACTION TO BE TAKEN ALONG WITH CLOSING FILE*** | |
| DFCLS\_ABORT | * Causes all changes, since file was opened OR last dfckp to be discarded. * It is used when file is opened with DBOPN\_DETAC. * If file is not opened with DBOPN\_DETAC, changes are already filed, so this is irrelevant |
| DFCLS\_COMMIT | * Causes all changes to be committed. * Ignored if used with file is not opened with DBOPN\_DETAC |
| DFCLS\_RELFC | * Releases sub-file and delete it from DASD. * All overflow block is released. * If prime is POOL, then prime block is released * If prime is FIXED, then prime block is initialized to empty * If sub-file is INDEXED, then sub-file is de-indexed from all of its indexes. |
| DFCLS\_RELEASE | Release SW00SR slot when sub-file is closed |
| * DFCLS\_NORELEASE | * Prevents SW00SR slot from being released when file is closed. * Used if we want to process **same** sub-file at a later time * Any key parameters you have defined are also retained. |
| * DFCLS\_REUSE | * Retains SW00SR slot of the file * Used if we want to process **another** sub-file at later time. * Any key parameters you have defined are also retained |
| * DFCLS\_ALL | * Close all the sub-file opened * Shouldn’t be used with DFCLS\_INCLUDE or DFCLS\_EXCLUDE |

**EXAMPLES**

* **Close a file and commit (file the changes made)**

dfcls ( file\_ptr, DFCLS\_COMMIT);

* **Close all the file**

dfcls (NULL,DFCLS\_ALL);

* **Abort the file modifications**

dfcls(file\_ptr,DFCLS\_ABORT);

## DBADD: ADD AN LREC

* The DBADD command adds a new LREC to a sub file,
* Facilitates addition of LREC BEFORE or AFTER the current LREC,
* At a position in the sub file as defined by keys and organization,
* By supplying a LRECNBR or at the end of the sub file

**PARAMETERS AND THEIR FUNCTION**

|  |  |
| --- | --- |
| **PARAMETER TO DESCRIBE WHICH SUB-FILE TO REFER** | |
| REF= ,  FILE | * This is used to identify the file to be accessed. * Detailed in Section 4.2 |
| ALG=  ORD=  FADDR= | * Provides the method use to specify particular Sub-file. Detailed in Section 4.3 * Area that contains the algorithm argument must not be modified and accessible until sub-file is closed and SW00SR is released. |
| **PARAMETER TO DESCRIBE ADDRESS OF NEW LREC TO BE ADDED** | |
| NULLLREC | * Adds empty LREC to sub file. * Specify the length of the empty LREC and primary key of empty LREC * Normally used with W-type and T-type files. * Adding NULLREC to UP/DOWN organized sub file might corrupt the organization. * DBMOD is used when any field of such LREC is modified |
| NEWLREC= | Specifies the address of new LREC to be added as arguments. |
| **PARAMETER TO SPECIFY ADDRESSIBILITY** | |
| SUFFIX= | Allows you to use the same DSECT to map two different areas of storage |
| REG= | Returns the address of the current LREC |
| **PARAMTERS TO DECIDE THE POSITION OF THE LREC TO BE ADDED** | |
| LRECNBR | * Sequence number of LREC at which added LREC should be placed * LREC will be added immediately after specified LREC number |
| FAST | * Generates inline code, can’t be used with KEYn Parameters. * Active keys get deactivated * Used only for Migration |
| AFTER  BEFORE | Decides whether to add LREC AFTER or BEFORE current LREC |
| UNIQUE | * Adds LREC only if there is no matching LREC previously exist in the sub file. * UNIQUE=YES can be coded in DBDEF to make it default option with all DBADDs for the file |
| KEY parameters | * This is use to specify position of the LREC and described in details in Section 4.4 * With each Key it is important to provide ORGANISATION and follow rules of |
| KEYLIST | it is used when we have to specify more than 6 keys. |
| UP|DOWN|  NOORG | Provides the organization of particular key to decide the position of the LREC |
| * **ADDITIONAL PARAMETER FOR INDEXING** | |
| INDEX = HOLD or NOHOLD | **BASIC INFORMATION ABOUT INDEX PARAMETER**   * Adds an LREC to a detail sub-file or intermediate index sub-file where the index structure does not yet exist. * If you specify this parameter, the algorithm that is defined for the new sub-file must be #TPFDBFF. * When you specify this parameter, the sub-file is created and indexed by adding an index LREC in the index file that references the sub-file.   **INFORMATION ABOUT INDEX= HOLD**   * Holds any index files that references the sub-file we are accessing * Holding occurs if Bits4 and 5 in &SW000OP2 global set symbol in DSECT macro   **INFORMATION ABOUT INDEX=NOHOLD**  Does not hold index files that reference the sub-files you are accessing |

EXAMPLE:

* **Add an LREC stored in EBW000**

DBADD REF=RRC1BR,NEWLREC=EBW000,UP

KEY1=(R=RRC1KEY,S=EBW002),

KEY2=(R=RRC1NAM,S=EBW003,L=L'RRC1NAM)

* **Add an empty LREC**

DBADD FILE=WRXXBR,PKY=#WRXXK10,REG=R5,

KEY1=(PKY=#WRXXK10),NULLREC==AL2(#WRXXL10),

ERROR=ADDERROR

* **Add an LREC just after the current LREC**

DBADD REF=RRC1BR,NEWLREC=EBW000,AFTER

* **Add an LREC just before the current LREC**

DBADD REF=RRC1BR,NEWLREC=EBW000,BEFORE

* **Add an LREC, address of the LREC is stored in the register**

DBADD REF=LW01SR,UP,REG=R2,

NEWLREC=R5, KEY1=(PKY=#LW01K50),

KEY2=(R=LW01SKY, S=LW01SKYX), ERROR=MSFADD1

* **Add an LREC and make sure that added LREC only if there is no matching LREC earlier**

DBADD REF=RRC1BR,NEWLREC=RRC1RECA,UP,UNIQUE,

KEY1=(R=RRC1KEY, S=#RRC1K80),

KEY2=(R=RRC1NAM, S=RRC1NAMA,L=L'RRC1NAM)

**NOTES:**

* When LRECNBR is added with the active keys, then only LRECs that match the key conditions are included in the sequence numbering.
* If UNIQUE = YES is coded in DBDEF macro for a file, all DBADD statements for that file default to UNIQUE
* Address of the newly added LREC is placed in SW00REC field of SW00SR Slot.
* If REG= parameter is used, then Register mentioned also contains this address.
* SW00RTN, #BIT1 is set if UNIQUE parameter was used and there was already a LREC present.

## dfadd–ADD ONE OR MORE LRECS TO A SUB-FILE

**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);

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER TO DESCRIBE WHICH SUB-FILE TO REFER** | |
| access | Method you want to use to access the sub file.  Use one of following values   * DFADD\_ALG * DFADD\_FADDR * DFADD\_FADDR8 * DFADD\_ORD |
| acc | Contains the value to be used with the access parameter to get method for accessing file |
| file | Pointer to the base address of the SW00SR slot of the file that you want to access |
| **PARAMETER TO DESCRIBE lrec AND WHERE TO ADD lrec** | |
| Options | * DFADD\_AFTER – Add LREC after current LREC * DFADD\_BEFORE – Add LREC before current LREC * DFADD\_FAST – Used for migration * DFADD\_INDEX –   + Add LREC to detail sub-file or intermediate index sub-files where index doesn’t occurs   + Sub-file is created and indexed by adding LREC in index file referencing the sub-file * DFADD\_INDEX\_HOLD   + All actions of DFADD\_INDEX are executed   + Holds any index files that hold any index file that refer sub-files accessed * DFADD\_NOKEY – Deactivate any currently active keys * DFADD\_UNIQUE – make sure that there is no duplicate LREC added * 0 - None of the processing options is used |
| Pky | * Primary key of the LREC that you are adding * Mandatory parameter to be used with DFADD\_NULLREC |
| Nbr | * It is variable containing the LREC sequence number in the sub-file * 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. * When you specify this parameter, the LREC is added immediately after the specified LREC. * If you specify this parameter for an LREC number that does not exist, the LREC is not added. |
| Rec | * It is a pointer to the LREC that you are adding. |
| rec\_type | Type of LREC that is being added. Can be either   * DFADD\_NEWLREC – Add a new LREC * DFADD\_NULLREC- |

**RETURN CONDITIONS**

A pointer to the LREC that was added is returned.

**Examples**

* The following example adds LREC to the current sub-file and make sure it is UNIQUE in the sub-file.

dfadd(pDI22->di22int.pSw00sr, // This is obtained from dfopn

DFADD\_NEWLREC,

DFADD\_UNIQUE,

pDI22->di22int.pLrec0); // This is pointer to LREC structure

if(DF\_NR(pDI22->di22int.pSw00sr)

{

iError = FOUND;

}

* The following example adds LREC to a sub file specified with an algorithm.

The new LREC is in pCIZM->cizmrec.

The algorithm argument is Algorithm

dft\_fil \* pSwCRZM;

char Algorithm[8] = “”;

memcpy(&Algorithm[6],

&pCIZM->cizmalg,

sizeof(pCIZM->cizmalg));

dfadd\_acc(pSwCRZM,

DFADD\_NEWLREC,

DFADD\_ALG,0,

&pCIZM->cizmrec,

Algorithm);

**NOTES**

* + Before using DBADD\_BEFORE or DBADD\_AFTER, current LREC must be established (using dfred)
  + When attempt is made to add an LREC when no sub file is defined and DFADD\_FADDR is coded, the z/TPFDF product obtains a prime block from pool and inserts the LREC into it.
  + Address of this prime block is stored in the SW00FAD field of the SW00SR slot and the record code check is stored into the SW00WCC field.

## DBDEL - DELETE AN LREC

* The DBDEL command deletes an LREC from a sub file.
* DBDEL can delete
  + the current LREC,
  + LREC(s) with matching KEYn combinations,
  + DBDEL can also delete an LREC by supplying a LRECNBR or a LRECNBR and KEYn combination
  + By default deletes current LREC

**PARAMETERS FOR DBDEL**

|  |  |
| --- | --- |
| **PARAMETER TO DESCRIBE FILE TO ACCESSED** | |
| REF=  FILE= | * This is used to identify the file to be accessed. * Detailed in Section 4.2 |
| ALG= |  ORD= |  FADDR= | * Provides the method use to specify particular Sub-file. Detailed in Section 4.3 * Area that contains the algorithm argument must not be modified and accessible until sub-file is closed and SW00SR is released. |
| **PARAMETER TO SPECIFY WHICH LREC TO DELETE** | |
| LRECNBR | Number of the LREC which we want to delete |
| LAST: | * To delete the last LREC in a sub-file, * Can also be used with KEYn parameters in which case the LAST matching LREC will be deleted |
| NEXT | * Deletes next LREC after current LREC |
| KEYn | Use to specify particular matching LREC, same as that used in DBRED |
| ALL | * All LRECs in the sub-file will be deleted if specified, * When used with the KEYn parameter, deletes all matching records. * If ALL records of sub-file are deleted and if the prime is a pool record, then all the prime and overflow files will be released back to the system. * If prime is Fixed, it is filed to Disk and its overflows are released |
| BEGIN: | Starts from the beginning of the sub-file before searching for LRECs to delete |
| UPWARD | * Deletes all LRECs from (not including) to FIRST LREC in the sub-file * Only Matching LRECs are deleted in KEYn parameter is specified |
| DOWNWARD | * Deletes all LRECs from (and including) current LREC to the LAST LREC in sub-file * Only matching LREC are deleted if KEYn parameter is specified |
| **PARAMETER TO SPECIFY WHAT TO DO WITH THE PROCESSING** | |
| REG= | * Register in which address of the next LREC following LREC to be deleted is returned. * This is also stored in SW00REC. |
| ERROR  ERRORA | * Processing branches to location specified in SPM label error mentioned in ERROR * Processing branches to location specified in Assembler label error mentioned in ERRORA |
| **PARAMETER TO HANDLE REFERENCED SUB-FILE** | |
| EXCLUDE | * Exclude specific sub-files that are referenced from the LREC that is being deleted. * Specified sub-files are not released * LIST =   list of sub-files to be excluded.  First 2 byte contains number of files in the list,  followed by list of IDs to be excluded   * ALL =   none of the sub-files referenced by LIST to be deleted will be released |
| INCLUDE | * Releases the sub-files that are referenced from the LREC that is being deleted.   LIST =  list of sub-files to be deleted  First 2 byte contains number of files in the list,  followed by list of IDs to be released  ALL =  All the sub-files referenced by LIST to be deleted will be released |

**EXAMPLE:**

* **DELETE RECORDS WITH KEY 10 AND MATCHING LNIATA**

DBDEL REF=AROZAE,ERROR=UGZ8E4,NOORG,

KEY1=(PKY=#AR0ZK10),

KEY2=(R=AROZTRM,S=EBROUT)

* **DELETE CURRENT LREC**

DBRED REF=RRC1BR,UP,REG=R4,ERROR=READERROR, make the

KEY1=(PKY=#RRC1K80), LREC pointer to point to the

KEY2=(R=RRC1NAM,S=MI0NAM) record to be deleted

DBDEL REF=RRC1BR delete the record

**NOTES: -**

* SW00REC contains pointer to next LREC in the sub-file (after the last deleted LREC)
* If ALL parameter is used, SW00REC contains 0 and SW00RTN is set to X’40’
* If you use the LIST parameter, check SW00RTN and bit 0 in SW00RT2.
* If you use the FULLFILE parameter, check the error count in the SW00RT1 field because the SW00RTN field only has an end-of-file indicator set.
* If you do not specify any optional parameters and a current LREC (usually the last LREC read) has been located, the DBDEL macro deletes the current LREC. Any previously specified keys are ignored.
* If you do not specify any optional parameters and a current LREC has not been located, the DBDEL macro result cannot be predicted.
* If deleted LREC contains index to other file, then
* You can use INCLUDE, EXCLUDE parameter to specify that DBDEL should also delete those referenced sub-file.
* Referenced sub-files are not de-indexed automatically.
* After an LREC is deleted,
* Current LREC position is no longer established.
* If you issue a z/TPFDF macro that relies on the current LREC position, the result cannot be predicted.
* To ensure that the current LREC position is established, you must issue a macro that does an explicit read (such as DBRED) or an implicit read (such as DBDEL with the NEXT parameter specified).

## dfdel– DELETE ONE OR MORE LRECs

**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);

|  |  |
| --- | --- |
| **PARAMETER TO SPECIFY THE SUB-FILE ACCESSED** | |
| file | Pointer to the base address of the SW00SR slot that you want to access |
| access | Method you want to use to access the sub-file. It can be either of the following   * DFDEL\_ALG * DFDEL\_FADDR * DFDEL\_FADDR8 * DFDEL\_ORD |
| acc | * is an ordinal number, a file address, or a pointer to an algorithm string that specifies the sub-file you want to access. * The type for this parameter is determined by the value you specify for the*access* parameter. |
| **PARAMETER TO MENTION WHICH LREC TO DELETE OR WHERE TO SEARCH** | |
| options | * Followings are the processing options for this functions * DFDEL\_ALL   Delete all the LRECs   * DFDEL\_ALL\_DOWNWARD   Delete all the LRECs from (and including) current LREC to last LREC.   * DFDEL\_ALL\_UPWARD   Delete all the LRECs from (but not including) current LREC to first LREC.   * DFDEL\_BEGIN   Start from the BEGIN when searching for the LRECs to be deleted   * DFDEL\_FULLFILE   Delete LRECs from every sub-file of the file.   * DFDEL\_LAST   Delete last matching LREC of the subfile |
| **PARAMETER TO MENTION WHAT SHOULD BE DONE WITH INDEXED SUB-FILE** | |
| List\_type | * Specifies which files to delete.   + DFDEL\_INCLUDE   + DFDEL\_EXCLUDE |

**RETURN CONDITIONS**

Pointer to the next LREC in the sub-file.

**EXAMPLES**

* **The following example deletes all LRECs in a sub-file.**

dfdel (pSw00sr, DFDEL\_ALL);

* **Following example deletes ALL LRECs of the sub-files mentioned in the list**

dfdel(pSw00sr,DFDEL\_INCLUDE,DFDEL\_ALL,pIDList);

Here **pIDList** is pointer to structure which contains following items

* + - number of item followed by
    - list of 2 byte IDs in Char or hex format.

## DBMOD -- MODIFY AN LREC

* The DBMOD command lets you to modify the fields in an existing LREC,
* It **indicates** that current LREC has got modified
* Modified LREC and the old copy of the LREC should be of the **same size**,
* Should also ensure that Key fields that are used for organization are not changed so as not to corrupt the organization
* Modification is applied to the current LREC.

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER SPECIFYING SUB-FILE TO BE ACCESSED** | |
| REF=  FILE= | These are the mandatory field to identify the file being referred |
| ALG= or  FADR= or  ORD= | Provides the method of accessing particular sub-file |
| **PARAMETER SPECIFYING FROM WHERE TO SEARCH AND WHICH** | |
| FULLFILE | Modifies the LRECs in all the sub-files of the file |
| ALL | Modifies all LREC in the open sub-file beginning with the current LREC |
| MODLIST | Specifies the base register of the modification key List  Details about Setting Keylist will be discussed in DBKEY |
| BEGIN | Search from the beginning of the sub-file for LREC to find LRECs to modify |
| REG= | Register in which address of the current LREC is returned. |

**EXAMPLES:**

* To modify record of a person with given name, after find it

// Initial read the record

DBRED REF=RRC1BR,UP,REG=R4,ERROR=READERROR,

KEY1=(PKY=#RRC1K80), find the record to be modified

KEY2=(R=RRC1NAM,S=MI0NAM)

#IF (DBFOUND,YES) if record found

#IF RRC1STA,NE,X'FF' make modifications

MVI RRC1STA,X'FF'

DBMOD REF=RRC1BR file the modified record

#ELSE

#GOTO ITEM\_ALREADY\_CANCELLED

#EIF

#ELSE

#GOTO PAX\_NOT\_BOOKED

#EIF

**NOTES:**

* At the end of DBMOD,
* SW00REC contains the address of the modified LREC if not updating all the LREC
* SW00REC = 0 and SW00RTN, #BIT1 (not found) or #5(EOF) if updating ALL LRECs
* Don’t 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.

* DBMOD macro sets an indicator in the block to say that it has been changed.
* The z/TPFDF product writes this block to DASD when you close or checkpoint the sub-file.
* There is no KEYLIST parameter with DBMOD, so use DBKEY before calling DBMOD with ALL parameter if you want to modify all records matching particular keys.

## dfmod– PERFORM OR INDICATE LREC MODIFICATIONS

**FORMAT**

dft\_rec \*dfmod (dft\_fil \*file);

dft\_rec \*dfmod\_all (dft\_fil \*file, dft\_kyl \*mod\_list);

dft\_rec \*dfmod\_all\_opt (dft\_fil \*file, dft\_opt options, dft\_kyl \*mod\_list);

dft\_rec \*dfmod\_all\_acc (dft\_fil \*file, dft\_opt options, dft\_kyl \*mod\_list, dft\_xxx acc);

dft\_rec \*dfmod\_all\_key (dft\_fil \*file, dft\_opt options, dft\_kyl \*mod\_list, dft\_kyl \*key\_list);

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER TO SPECIFY SUB-FILE TO BE ACCESSED** | |
| file | Pointer to the base address of SW00SR slot of the file |
| acc | Parameter to specify particular sub-file to access   * ALG * FADDR * ORD |
| **PARAMETER TO SPECIFY WHICH LRECs TO BE MODIFIED** | |
| key\_list | Pointer to Selection Key that determines which LREC to be modified. |
| options | Options to specify how to   * DFMOD\_BEGIN -Search from beginning for the LREC to be modified * DFMOD\_FULLFILE- Modifies LRECs in all sub-files of the file * DFMOD\_NOKEY – Deactivates any current active keys * 0 = Specifies no processing option will be used |
| **PARAMETER TO SPECIFY HOW LRECs ARE MODIFIED** | |
| mod\_list | A pointer to the modification key list that describes how the LRECs are to be modified. |

**RETURN CONDITIONS**

* 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.
* After FULLFILE modification, value returned is 0 and SW00RTN contains #DBEOF (Bit5 ON)

**EXAMPLES**

dfmod(pSw00sr);

## DBRET - RETAIN LREC

* This macro retains the file address and displacement in block of the current LREC.
* This LREC can be later read using DBRED macro with PREVIOUS with STACK, or STACKREF

**PARAMETER**

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= and/or  FILE= | File that we want to access |
| REG= | Register in which address of current LREC (also contained in SW00REC) will be returned |
| STACK= stckloc | Retains the LREC  where **stckloc** is location of 10-byte field that contains the details about the LREC |
| STACKREF= | Retains an LREC where value is assigned to LREC.  It can be either   * 4-byte value * Register containing 4-byte value * Absolute value that doesn’t exceed 4-bytes |
| ERROR= or ERRORA= | * Processing branches to ERROR (SPM label) if serious error is detected * Processing branches to ERRORA (Assembler label) if serious error is detected. |

**NOTES:-**

* + To ensure that an LREC is retrieved correctly adhere to following rules when using retained LRECs

|  |  |
| --- | --- |
| * STACK is specified OR   Neither STACK nor STACKREF is specified | * STACKREF can’t be specified on any subsequent DBRED or DBRET macro call. * File must be closed and reopen file first |
| * IF you specify STACKREF parameter | * This parameter must be specified on all macro calls to the same open file * File must be closed and reopen the file if you want to specify STACK parameter Or we don’t want to specify retained LREC parameter |

* If you do not specify the STACK or STACKREF parameter, the details of only the current LREC are retained.
* Each entry control block (ECB) can retain a maximum of 250 unique STACKREF parameter values containing position information in one or more files.
* When a file is closed, all retained positions for that file are deleted.

**EXAMPLE**

DBRET REF=CR90KL, STACKREF ==F’5’ // Retain Current LREC

DBRED REF=CR90KL, STACKREF ==F’5’ // Read back retained LREC

## dfret– RETAIN A LOGICAL RECORD POSITION

**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);

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER TO ACCESS SUB-FILE** | |
| file | A pointer to the base address of the SW00SR slot (defined in c\_sw00sr.h) of the file that you want to access |
| **PARAMETER TO SPECIFY OPTIONS (options)** | |
| * DFRET\_CURRENT   Specifies that we want to retain current LREC   * 0   No Processing Option is required | |
| **PARAMETER TO SPECIFY STACK TYPE (stk\_typ)** | |
| DFRET\_STACK | * Pointer to 10-byte stack area is specified in stk * Wherever possible use DFRET\_STACKREF option |
| DFRET\_STACKREF | * Specifies that stack reference number is specified in stk |
| **PARAMETER TO SPECIFY STACK REFERENCE (stk)** | |
| Stack reference number | * Used with DFRET\_STACKREF, it is reference number used to identify stack number * If it is not unique, then current LREC information will replace LREC already associated with this reference number |
| Stack | Pointer to Stack area used with DFRET\_STACK |

EXAMPLE

dfret\_stk(file\_ptr,DFREF\_CURRENT,DFRET\_STACKREF,100);

## DBREP : REPLACE AN LREC

* The DBREP is used to replace the current LREC with a new LREC that is supplied,
* Similar to DBMOD
* the difference being that the size of the new LREC can be different from the old LREC,
* The Key fields used for organization however should be the same
* If the key fields are to be changed, one should DELETE the old LREC and ADD the new LREC, so that the new LREC finds its place according to the file organization.

**PARAMETERS**

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= and/or  FILE= | Specifies the file that you want to access, |
| NEWLREC= | It contains the address of LREC which will replace the current LREC |
| ERROR= or ERRORA= | * Processing branches to ERROR (SPM label) if serious error is detected * Processing branches to ERRORA (Assembler label) if serious error is detected. |
| REG= | * In this address of the current LREC is stored. |

**EXAMPLE:**

DBRED REF=RRC1BR,UP,REG=R4,ERROR=READERROR,

KEY1=(PKY=#RRC1K80), find the LREC to be modified

KEY2=(R=RRC1NAM,S=MI0NAM)

#IF (DBFOUND,YES) if record found

#IF RRC1STA,EQ,X'20'

MVC EBW000(#RRC1L80),RRC1SIZ

MVC EBW000+#RRC1L80(L'EXPLOSIVE),EXPLOSIVE

LH R6,EBW000 build the new LREC

LA R6,L'EXPLOSIVE(,R6)

STH R6,EBW000

DBREP REF=RRC1BR,NEWLREC=EBW000 replace the LREC

#ELSE

#GOTO ITEM\_NOT\_EXPLOSIVE

#EIF

#ELSE

#GOTO NAM\_NOT\_BOOKED

#EIF

EXPLOSIVE DC C'ITEM IS DANGEROUS GOODS'

**NOTE AND RETURN VALUE:**

* DBREP replaces current LREC and doesn’t perform DBRED, So, DBRED should be performed before calling DBREP to locate the current LREC.
* New LREC can be smaller or larger or equal to the size of the current LREC.
* To preserve organization of the file, never replace key fields.

## dfrep–REPLACE A LOGICAL RECORD WITH ANOTHER LOGICAL RECORD

**FORMAT**

dft\_rec \*dfrep(dft\_file \*file, dft\_rec \*rcd);

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| file | A pointer to the base address of the SW00SR slot of the file that you want to access |
| rcd | Pointer to replacement LREC |

**RETURN CONDITIONS**

A pointer to the new LREC.

**EXAMPLE**

The following example replaces an existing LREC with the LREC pointed to by rec\_ptr.

dft\_file \*file\_ptr;

dfrep(file\_ptr,&rec\_ptr);

# MISCELLANEOUS GENERAL MACROS

The TPFDF Command macros that are discussed in this section are

* DBCRE
* DBADR
* DBDSP
* DBKEY
* DBSETK
* DBSPA
* DBIFB
* DBFRL



## DBCRE COMMAND MACRO:

* This command creates a sub file, empty pool sub file or empty indexed pool sub-file with its corresponding index file.
* Performs the GETFC function internally,
* It also sets up the SW00SR slot for the new sub-file,
* The pool type depends upon the DSECT name.
* The size of the pool file is determined from the DSECT/DBDEF.

|  |  |
| --- | --- |
| **PARAMETERS** | **FUNCTION** |
| REF= or/And  File= | Specifies the file that you want to access, |
| ALG= | Identifies the sub file that we want to access |
| INDEX | * It creates an indexed sub file and * Inserts index LREC referencing this sub file 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 sub file must be #TPFDBFF. |
| REG= | * Register in which address of the header of the prime block of the created sub-file is returned |
| ERROR = Or  ERRORA = | * Branches to specified location if serious error is detected where ERROR is SPM label * Branches to specified location if serious error is detected where ERRORA is Assembler Label |

**RETURN VALUE:-**

* The address of the header of the prime block of the sub-file that is created is placed in the SW00REC field of the SW00SR slot.
* If you specify the REG parameter, this address is placed in the specified register.

**EXAMPLE:**

* **CREATE EMPTY FILE AND STORE ADDRESS TO A STORAGE**

DBCRE REF=QY9REF,FILE=CR90XX AND CREATE AN EMPTY FILE

MVC QY9CPC,SW00FAD FILE ADDRESS TO INTERFACE

## dfcre–CREATE A SUB-FILE

**FORMAT**

dft\_hdr \*dfcre (dft\_fil \*file, dft\_opt options);

dft\_hdr \*dfcre\_chka (dft\_fil \*file, dft\_opt options);

dft\_hdr \*dfcre\_alg (dft\_fil \*file, dft\_opt options, dft\_alg \*alg);

|  |  |
| --- | --- |
| **PARAMETER TO SPECIFY SUB-FILE TO BE ACCESSED** | |
| file | Pointer to the base address of SW00SR slot of the file |
| alg | A pointer to an algorithm argument that identifies the subfile. |
| **PARAMETER FOR NEWLY CREATED SUB-FILE** | |
| rcc | User defined RCC used in created sub-file |
| options | Processing options   * DFCRE\_INDEX   + Creates an indexed sub file and   + inserts an index LREC referencing this sub file into the related indexed files * 0 = Specifies no processing option will be used |
| ERROR = Or  ERRORA = | * Branches to specified location if serious error is detected where ERROR is SPM label * Branches to specified location if serious error is detected where ERRORA is Assembler Label |

**RETURN CONDITIONS**

A pointer to the main storage address of the header of the prime block of the created sub file is returned.

**EXAMPLES**

dfcre(pSw00SR,0);

## DBADR: PROVIDE FILE ADDRESS OF A PRIME BLOCK.

* This command allows you to find a file address and ordinal number of a prime block in the fixed file
* This is also used to specify a range of ordinals to be used in subsequent FULLFILE processing

**PARAMETERS FOR DBADR**

|  |  |
| --- | --- |
| REF=  FILE= | Specifies the file that we want to access |
| ALG= |  ORD= | Specifies the Algorithm or Ordinal number, which is used to specify the correct sub file. |
| BEGALG or BEGORD | * **BEGALG** helps in calculating the beginning ordinal number to be used in macros that specified the FULLFILE parameter. * **BEGORD** specifies the beginning ordinal number |
| ENDALG or ENDORD | * **ENDALG** helps in calculating the ending ordinal number to be used in macros that specified the FULLFILE parameter. * **ENDORD** specifies the beginning ordinal number |
| WRAPAROUND | * Reads LREC from start of the file to the end until it has read the whole file. * Use this when you intend to use full file processing |
| ERROR = Or  ERRORA = | * Branches to specified location if serious error is detected where ERROR is SPM label * Branches to specified location if serious error is detected where ERRORA is Assembler Label |

**ENTRY REQUIREMENT**

* If you specify the BEGORD or ENDORD parameter without specifying the ORD or ALG parameter, you must specify the ORD or ALG parameter on the previous and corresponding DBOPN macro call or a system error will occur.

**RETURN CONDITION**

* IF ALG or ORD is specified,
  + SW00WR1 contains the file address of the corresponding prime block.
  + SW00WR2 contains the ordinal number of the corresponding prime block
* If BEGORD or BEGALG parameters are specified,
  + SW00ORD contains the ordinal number of the corresponding prime block.

ELSE

* + SW00ORD=0 and FULLFILE processing starts with first ordinal number in file
* If ENDORD or ENDALG parameters are specified,
  + SW00END contains the ordinal number of the corresponding prime block.

ELSE

* + ENDORD=0 and FULLFILE processing ends with last ordinal number in file
* When you use this macro, subsequent FULLFILE processing occurs in the ordinal range SW00ORD–SW00END.

EXAMPLE:

* DBADR REF=RRC1BR,BEGALG=='A',ENDALG=='O' range of sub-file

DBRED REF=RRC1BR,FULLFILE .. THIS CAN GO ON IN A LOOP ...

* DBADR REF=RR10SR,BEGALG=EBX082,ENDALG=EBX070,WRAPAROUND

## dfadr–PROVIDE THE FILE ADDRESS OF A PRIME BLOCK

* This function is used to get the file address and ordinal number 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, dft\_opt options, dft\_alg \*alg);

void dfadr\_ord (dft\_fil \*file, dft\_opt options, dft\_ord \*ord);

void dfadr\_beg\_ord (dft\_fil \*file, dft\_opt options, dft\_alg \*beg ,dft\_alg \*end);

**PARAMETERS AND THEIR FUNCTION**

|  |  |
| --- | --- |
| file | File Pointer to the base address of the SW00SR slot that we want to access |
| alg  ord | * Pointer to algorithm argument that identifies the sub file. * Ordinal number of the sub file that identifies the sub file |
| beg | * Pointer to algorithm argument that is used to calculate begin ordinal of the file. This is used as starting ordinal during the full file processing |
| end | * Pointer to algorithm argument that is used to calculate ending ordinal of the file. This is used as end ordinal during the full file processing |
| option | Following are the processing options for this function.   * DFADR\_NODUMP (Don’t issue any of the following dumps   DB0100, DB0102, DB0117, DB0123, DB0138, DB0140   * DFADR\_WRAPAROUND (reads LRECs from the start of the file to the end until it has read the whole file.) * 0 |

**RETURN CONDITIONS**

* 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.

|  |  |
| --- | --- |
| **PARAMETER USED** | **RETURNED VALUE** |
| * ALG= or ORD= | * SW00WR1 contains the file address of corresponding prime block * SW00WR2 contains the ordinal number of that prime block |
| * BEG parameter | * SW00ORD is set to ordinal number of the corresponding prime block. * If BEG parameter is not provided, SW00ORD = 0 and subsequent fullfile processing will start with first ordinal in the file |
| * END parameter | * SW00END is set to ordinal number of the corresponding prime block. * If END parameter is not provided, SW00END = 0 and subsequent fullfile processing will end with the last sordinal in the file |

Examples

dft\_fil \*file\_ptr;

dft\_ord Ordinal = 3;

dfadr\_ord(file\_ptr,0,Ordinal); // find address of 3rd ordinal

## DBDSP COMMAND MACRO:

* This command allows sending output to a terminal, selected LRECS from an opened sub file;
* LRECs must be made of valid printable EBCDIC characters

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= or /and  FILE= | Used to specify the file that we want to access |
| ALG or FADDR or ORD | Used to identify sub file that needs to be accessed |
| *KEYn(R=,S=|D=|M=,L=,*  *C=.UP|DOWN|NOORG) (n=1..6)* | It is used to provide parameters for searching LREC to be displayed |
| *ERROR|ERRORA* | * ERROR is SPM label to which processing branches if any serious error is detected when processing this macro * ERRORA is assembler label to which processing branches if any serious error is detected when processing this macro |
| FULLFILE | * Used for allowing display LRECs from whole file instead of just one sub-file * Don’t use this parameter with W-type file or NOCLOSE option |
| NOCLOSE | * Specifies that sub file should not be closed after display * Control returns to the program after display |
| NOFINAL | * Let’s system know that is only part of the display |
| NOUIO | * Returns control back to application after display |
| RELFC | * Closed sub file is released |
| STRIP | Data that need not be displayed.  2 byte size field is automatically discarded so no need to include that  Do not use R14 or R15 with the STRIP parameter  It can be either register containing number of bytes to be discarded OR 2-byte label containing that value |
| WTOPC | IF YES,   * LREC is displayed in WTOPC format. * Maximum length displayed for LREC is 255 bytes and * LONGTERM, NOUIO are ignored.   IF NO   * OMSG format will be used. |

**EXAMPLE**:

DBDSP REF=WRK2BR, KEY1=(PKY=#WRK2K20),

STRIP==AL2 (WRK2NAM-WRK2KEY)

DBDSP REF=IWDWDF,STRIP==AL2(L'IWDWKEY),ERROR=IWD-ERR,

OPMT=DSPOPMT

DBDSP REF=GRM7KL,NOUIO,NOKEY,STRIP==AL2(L'GRM7KEY),

ALG=GWERALG

**NOTES**

* If WTOPC = NO (default), OR YES value is not specified Or &ACPDBAA =0

Data level 1 (D1) and data level 3 (D3) are not data level independent (DLI)

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.
* The sub-file 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.
* If the AUTODEINDEX=YES parameter is specified on the DBDEF macro and an indexed sub-file is empty when the sub-file is displayed, the sub-file will be de-indexed from all its indexes and all corresponding pool file addresses will be released.

## dfdsp–DISPLAY LRECS FROM A SUB-FILE

**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);

**PARAMETER**

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| File | * Pointer to base address of the SW00SR slot of the file that we want to access * If acc parameter is not specified, last sub-file associated with this SW00SR slot will be accessed |
| Access | * Method used to access Sub file. It can be one of the following values.   + DFDSP\_ALG   + DFDSP\_FADDR   + DFDSP\_ORD |
| Options | * These are processing options for this function. * One of the following values are used * DFDSP\_FULLFILE   + LREC from whole file can be displayed.   + Don’t use this value with W-type file or DFDSP\_NOCLOSE option * DFDSP\_LONGTERM   + Instructs application program to prepare OMSG using long-term pool file.   + else short-term pool files are used * DFDSP\_NOCLOSE   + Indicate 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 option specified. |
| str | * Is a variable containing the number of bytes that you want to strip from the start of each LREC |

**EXAMPLES**

The following example displays a message consisting of all the LRECs from a sub-file pointed to in file\_ptr. The sub-file is released after use.

dft\_fil \*file\_ptr;

dfdsp (file\_ptr, DFDSP\_RELFC);

## THE DBKEY COMMAND MACRO:

* This command macro allows you to define a list of keys for use with a sub file
* This key list is used by subsequent macros that access specified sub file.
* These keys can then be used with DBRED, DBADD, DBDEL & DBMOD using the KEYLIST parameter

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTIONS** |
| REF= or FILE= | Refers to the file that we are accessing |
| KEYLIST | It can be either Register or label or Address of label which holds the address of the keylist |

**STEPS TO ACTIVATE KEY LIST**

* DBSETK sets the following parameters to indicate the dynamic parameters to be use during the search process.
* DBKEY activates that key list in the SW00SR block.
* We can set up to 180 keys using this MACRO.

The keylist should have been set up previously (in DBSETK) following the format given below,

**Described by DSECT SW01SR**

SW01NKY DS H NUMBER OF KEY PARAMETERS DEFINED (MAX 6)

SW01KIT DS 0XL12 KEY ITEM LENGTH

SW01DIS DS AL2 DISPLACEMENT (R=.....)

SW01LEN DS AL2 LENGTH (L=.....)

SW01CON DS XL1 COMPLEMENT OF REQUIRED CONDITION CODE IN

\* BRANCH INSTRUCTION EG

\* EQ=X'70',NE=X'80',GT=X'D0',NH=X'20',

\* LT=X'B0',GE=X'40',

\* Z =X'70',O =X'E0',M =X'B0',NZ=X'80',

\* NM=X'40',NO=X'10'

SW01MSK DS XL1 TM INSTRUCTION IF NE X'00' MASK ELSE CLC

SW01SEA DS AL4 CORE ADDRESS WHERE SEARCH ARGUMENT IS

\* NOT USED IF SW01MSK IS SET

SW01ID1 DS XL1 OPTION INDICATORS

\* BIT 0 = 1 DOWN ORGANISED

\* BIT 1 = 1 UP ORGANISED

\* BIT 2 = 1 SPARE

\* BIT 3 = 1 2ND BASE

\* BIT 4 = 1 CLI WITH MASK IN SW01MSK

\* MASK IS RIGHTADJ. IN SW01FDS

\* BIT 5 = 1 CP INSTRUCTION

\* BIT 6 = 1 TM INSTRUCTION

\* MASK IS RIGHTADJ. IN SW01FDS

\* BIT 7 = 1 RESERVED (NOT USED HERE)

SW01ID2 DS XL1 SPARE BYTE

ORG SW01NKY

**EXAMPLE**

(CHECK EXAMPLE OF DBSETK)

## dfkey–ACTIVATE A KEY LIST

**FORMAT**

void dfkey(dft\_fil \*file, dft\_kyl \*keylist);

void dfkey\_nbr(dft\_fil \*file, dft\_kyl \*keylist,short int n);

|  |  |
| --- | --- |
| **FUNCTION** | **PARAMETER** |
| File | Pointer to base address of the SW00SR slot of the file that we want to access |
| Key\_list | Pointer to key list |
| N | Number of keys that we want to setup |

* DFKEY is used in 3 steps
  + we will set number of Keys using ***df\_nbrkeys***function
  + Then we set keys and condition using ***df\_setkey***function
  + Finally we activate the key list, using ***dfkey*** function.
* Then we use this KEYLIST in the functions.

**EXAMPLES**

**Setting up a key list with less than six keys using dfkey parameters**

**STEP 1:** Setting number of Keys using *df\_nbrkeys*

* + df\_nbrkeys(&Keys,1);

**STEP 2:** Set Keys and Conditions

* + df\_setkey(&keys, 1, offsetoff(struct wr5rbr, wr5rkey), 1, DF\_EQ, &pky, 0, DF\_NOORG, DF\_CHAR);

**STEP 3:** Activate the key-list

* + dfkey (file\_ptr, &Keys);

**STEP 4**: Use the key list in the function

* + dfred (file\_ptr, DFRED\_BEGIN)

## The DBSETK Command Macro:

* This command macro can be used to set up a key list without using the SW01SR macro,
* It allows you to define a list of keys for use with a sub file,
* Up to 180 keys can be specified.

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| BASE | Register that contains the base address of the keylist |
| KEYNUM | Number of Keys to be set up (1-180) |
| DIS | Displacement of the key field from the start of LREC.  IT can be either LABEL or REGISTER of Immeidate value (I/2) |
| LEN | Length of the key field. In the LREC .  It can be either LABEL (Eg- LEN=EBW000) OR Register (R4) or Immediate Value (I/1) |
| MSK | 1 Byte Mask Value to be used for search |
| SEA | Specify a search argument. |
| CON= | Specifies a condition for the match to be successful. IT depends on Value of ID1.  IF ID1= DF\_CONST, DF\_CHAR or DF\_MIXED, CON can be one of the following   * DF\_GT = TRUE if LREC field is greater than search argument * DF\_GE = TRUE if LREC field is greater or equal to search argument * DF\_EQ = TRUE if LREC field is equal to search argument * DF\_NE = TRUE if LREC field is not equal to search argument. * DF\_LE = TRUE if LREC field is less than or equal to search argument. * DF\_LT = TRUE if LREC field is less than search argument   IF ID1= DF\_MASK, then use one of the following   * DF\_NM = TRUE if selected bits are all zero or all ones * DF\_NO = TRUE if selected bits are **NOT ALL ONES**. * DF\_Z = TRUE if selected bits are all zeros. * DF\_NZ = TRUE if selected bits are not all zeros. * DF\_M = TRUE if selected bits are mixed ones and zeros. * DF\_O = TRUE if selected bits are all ONEs. |
| ID1 = | Option Byes and can be either of them   * DF\_DOWN = Indicate SUB FILE is DOWN organized on this key * DF\_UP = Indicate SUB FILE is UP organized on this key. * DF\_KEYS = Used if we want to use default keys * DF\_CONST = Indicate that MSK parameter contains 1-byte search argument * DF\_PACKED = Indicate that search argument is variable-length packed string * DF\_MASK = Indicate that MSK contains 1 byte mask * DF\_NOORG = Sub file is not organized on this field * DF\_CHAR = search argument is variable-length character string |
| ID2 | It contains Boolean connector   * DF\_OR = OR connection between this and next key conditions * DF\_ORIF = ORIF connection between this and next key conditions * DF\_AND = AND connection between this and next key condition * DF\_ANDIF = ANDIF connection between this and next key condition |

**EXAMPLE:**

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=#RRC1K80, \*

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=RRC1BR,KEYLIST=EBX000 ACTIVATE KEY LIST

DBRED REF=RRC1BR

* 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.

## df\_setkey– SETTING UP KEY IN KEYLIST

Purpose of this function is same as that of DBSETK macro

**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\_mod (dft\_kyl \*key\_list, short int nbr, short int dsp, short int len, char \*sea, char msk, char oper);

void **df\_setkey\_bool**(dft\_kyl \**key\_list*, short int *nbr*, short int *dsp*, short int *len*, char \**sea*,char msk, char bool);

|  |  |
| --- | --- |
| **PARAMETERS** | **FUNCTION** |
| key\_list | Pointer to Key list that will be used to setup active key list |
| nbr | Key number that we are setting up |
| dsp | * displacement of the key in the LREC * (offsetof (struct, field) can be used to find that |
| len | length of the key field in the LREC |
| Con | Condition that must exist for search to be successful  IF typ = DF\_CONST or DF\_CHAR or DF\_PACKED, then CON will be   * DF\_EQ = LREC key field is equal to search argument * DF\_NE * DF\_GT * DF\_LT * DF\_LE, * DF\_GE,   IF typ = DF\_MSK, theCON will be one of the following   * DF\_Z * **DF\_O** * **DF\_M** * **DF\_NZ** * **DF\_NM** * **DF\_NO** |
| Sea | It contains address of the search argument.  Set it to Zero while using msk parameter |
| Msk | It contains the either 1-byte search argument or 1-byte mask |
| ORG | Specifies the organization of key field. It can be either   * DF\_UPORG - means sub-file is UP organized on this key field * DF\_DOWNORG * DF\_NOORG |
| Typ | Type of the search argument that we are specifying.  Following values are used to specify what is there in msk parameters   * DF\_CONSTANT means msk parameter contains 1-byte search argument * DF\_MASK means msk parameter contains 1-byte mask   Following value specify the value specified by sea   * DF\_CHAR means search argument is variable-length character string * DF\_PACKED means search argument is variable-length packed decimal string |
| Oper | Indicate operation to be performed during global modification of LRECs  For all the instruction  SW01MSK/SW00SEA contains the source address  SW01DIS contains the displacement of the target  SW01LEN contains length for the operation   * **DF\_MVI** moves values contained in SW01MSK into LREC at displacement SW0DIS. * **DF\_MVC** moves character string whose address is SW01SEA into LREC (Displacement stored in SW01DIS) for length SW01LEN * **DF\_FILL** propagates the character contained in SW01MSK into LREC, starting at displacement given in SW01DIS for length SW01LEN   **Other OPER options are**  **DF\_OI**  OI Field at displacement specified in SW01DIS of LREC with content of SW01MSK   * **DF\_OC** = OC operation is Performed where   SW01DIS contains displacement of 1st Operand in the LREC  SW01LEN contains Length of Data to be ORed  SW01SEA contains Address of 2nd Operand   * **DF\_NI**   NI field at displacement SW01DIS with contents of SW01MSK   * **DF\_NC** = NC operation is Performed where   SW01DIS contains displacement of 1st Operand in the LREC  SW01LEN contains Length of Data to be ORed  SW01SEA contains Address of 2nd Operand   * **DF\_ADD** = Adds Fullword value whose address is in SW01SEA to the fullword value in LREC whose displacement is mentioned in SW01DIS |
| Bool | Type of Boolean connector between current and subsequent Boolean slots.  It can be   * DF\_OR * DF\_ORIF * DF\_AND * DF\_ANDIF |

**EXAMPLES**

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.

df\_keyp(&KeysQMB, DF\_EQ, DR01KA0, DF\_UPORG);

iKeyNumber = 1;

If (pIQMB->di01pfx)

{

iKeyNumber++;

/\* Find LREC with Matching PFX \*/

df\_setkey(&KeysQMB, iKeyNumber, offsetof(DR01RA0,dr01pfx),

member\_size(DR01RA0,dr01pfx), DF\_EQ,

&pIQMB->di01pfx, 0, DF\_UPORG, DF\_CHAR);

}

If(pIQMB->di01awb)

{

iKeyNumber++;

df\_setkey(&KeysQMB, iKeyNumber, offsetof(DR01RA0,dr01awb),

member\_size(DR01RA0,dr01awb), DF\_EQ,

&pIQMB->di01awb, 0, DF\_UPORG, DF\_CHAR);

}

df\_nbrkeys(&KeysQMB,iKeyNumber);

dfkey (pIQMB->di01int.pSw00sr,&KeyQMB);

## The DBSPA Command Macro:

* This command macro allows you to get work space in the TPFDF interface block,
* It obtains and initializes workspace linked to SW00SR slot for the sub-file.

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= and/or  FILE= | Parameter to specify the file which will be accessed |
| SPACE OR  SPACEB OR  SPACEF | * SPACE = Allocates the workspace and initialize it to X’00’ * SPACEB = Allocates the workspace and initialize it to X’00’ * SPACEF = Allocates the workspace and intialise it to character provided.   Values to be provided with this parameters   * Number of bytes of space required. Max – 4069 bytes * Register which will contain base address of allocated space * Character with this space should be initialized (used with SPACEF only) |
| ERROR or ERRORA | * SPM Label (ERROR) or Assembler label (ERRORA) to which processing branches if there is any serious error encountered. |

**PROGRAMMING NOTE AND RETURN CODE**

* If number of space requested by SPACE or SPACEB or SPACEF is zero, then the space allocated previously will be released
* SW00WKA will contain the address of the work space allocated

**EXAMPLE:**

* Allocate the length = L’FIELD and save base address of the space in R5

DBSPA REF=RRC1BR, SPACEB = (#RRC1L80, R5)

* Release the previously allocated work space

DBSPA REF=RRC1BR,SPACE=(0)

* Allocate and Initialize the allocate 500 bytes with X’FF’

DBSPA REF=RRC1BR,SPACEF=(500,R5,X’FF’)

## dfspa–Create work space

This C function also performs same function as DBSPA performs.

**FORMAT**

void \*dfspa (dft\_fil \*file, dft\_spc spc, dft\_sps sps);

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| file | * Is a pointer to the base address of the SW00SR slot of the file that you want to access and * It is returned by the ***dfifb*** or ***dfopn***function |
| spc | * This provides the character that you want to use to initialize the workspace |
| sps | * This provides the size of the space, which can be a maximum of 4069 bytes. |

**RETURN CONDITIONS**

The address of the space that the z/TPFDF product has provided is stored in SW00WKA.

**Examples**

The following example creates a 400-byte area initialized with X’00’s.

dft\_fil \*file\_ptr;

dfspa (file\_ptr,0x00,400)

## DBIFB: CHECK SW00SR SLOT

* This command macro is used to check if a sub-file is open and
* if opened returns the SW00SR slot base address

|  |  |
| --- | --- |
| **PARAMETERS** | **FUNCTIONS** |
| FIRST or  NEXT or  REF = or  FILE = | * Purpose of REF = or FILE = is same as used with previous commands * FIRST is used to find the SW00SR details of the first slot with open sub-file * NEXT is used to find the SW00SR details of the next slot with open sub-file |
| NEWREF | Changes the reference of the file specified with REF= parameter |
| REG= | Specifies the register in which address of Current LREC of the target SW00SR slot will be returned |
| ERROR or ERRORA | SPM (ERROR) or ASSEMBLER (ERRORA) label to which processing branches if serious errors are found during execution of this MACRO |

**RETURN CODE AND PROGRAMMING NOTE**

* R3 contains the address of the SW00SR slot.
* We can check if particular file is opened or not, by checking R3 value
* R14 contains the 8-Byte location containing the reference name of the file when FIRST or NEXT parameter is used.

**EXAMPLE:**

* Following Code checks if FILE is opened and closed it if open

DBIFB REF=WRMBBR is file opened?

#IF (R3,NZ) yes...

DBCLS REF=WRMBBR,RELEASE close file/release

#EIF

* Following code check if FILE is opened and OPENs it if not already OPENED

DBIFB REF=LWS0SR is file opened?

#IF LTR,R3,R3,Z no....

DBOPN REF=LWS0SR,DETAC, open file

SPACE=(=Y(WD0PEVA-LWS0REC),R1)

#EIF

## dfifb–CHECK A SW00SR SLOT

Performs same function as DBIFB

**FORMAT**

* dft\_fil \*dfifb\_fst();

Return address of first SW00SR slot with an open subfile

* dft\_fil \*dfifb\_nxt();

Return address of SW00SR slot of next open subfile

* dft\_fil \*dfifb\_ref (dft\_ref \*ref);
* dft\_fil \*dfifb\_net (dft\_fil \*file, dft\_ref \*new);

|  |  |
| --- | --- |
| **PARAMETERS** | **FUNCTIONS** |
| ref | * Pointer to reference name of sub-file * Null-terminated if less than 8-bytes |
| file | Pointer to the base address of SW00SR slot |
| new | * Is a pointer to the character string (up to 8 bytes) you want to use as the new file reference name * Truncated with Blank if less than 8-bytes |

**RETURN CONDITIONS**

The address of the SW00SR slot.

**EXAMPLES**

* The following example checks if a sub-file has been opened.

if(dfifb\_ref(“CR90BR “) != NULL)

* The following example returns a pointer to the SW00SR slot of the first open sub-file.

fptr = dfifb\_fst();

## DBFRL MACRO : ENSURE AN ECB DATA LEVEL IS FREE

* This command macro is used to free a data level used by z/TPFDF for use by the Application
* After execution of this MACRO, z/TPFDF program will not be using this level in this ECB and TPF programs are free to use this level.

**PARAMETER FOR DBFRL**

LEVEL = hex or Dhex

* Either **hex** between X’0’ or ‘F’ to indicate the level to be made free OR level number preceded by D to indicate that we are providing Data Level.

**EXAMPLE:**

DBFRL LEVEL=DC free data level DC

## 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);

Where

*lev*

Is the data level you want to free, which can be in the range D0–DF.

**RETURN CONDITIONS**

The specified data level is freed.

**EXAMPLES**

The following example frees level DB.

1. Dffrl\_lev(DB);

## DBCKP MACRO: CHECKPOINT A SUBFILE

* The DBCKP command macro allows to checkpoint to database updates made to a TPFDF File ,
* TPFDF files all the blocks of the sub-file to DISK (issues the FILNC Macro).
* The current LREC pointer remains unchanged

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF = or FILE= | Specifies the file that needs to be referenced |
| DETAC or NODET | * DETAC places the file in detac mode after check-pointing the sub-file. Changes are not written to DASD until sub file is check=pointed again or closed. * NODET is used when sub file should not be placed in detac mode after it has been check pointed * This parameter cannot be used with W-type files. |
| *ERROR|ERRORA* | * ERROR is SPM label to which processing branches if any serious error is detected when processing this macro * ERRORA is assembler label to which processing branches if any serious error is detected when processing this macro |

**EXAMPLES:**

DBCKP REF=RRC1BR, REG=R4

DBCKP REF=RRC1BR, REG=R4, DETAC, ERROR=DBERROR

## dfckp–CHECKPOINT A SUB-FILE

**FORMAT**

dft\_rec \*dfckp (dft\_fil \*file, dft\_opt options);

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| file | Is a pointer to the base address of the SW00SR slot of the file that we want to access |
| Options | Are the processing options for this funct   * DFCKP\_DETAC   + Sub-file is placed in DETAC mode after checkpointing.   + All accessed blocks are saved in main storage.   + Any changes made to the LRECs are not written to DASD until the sub-file is check pointed 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.   + This option can’t be used with W-type files   DFCKP\_NODET   * + Sub-file is not placed in DETAC mode after checkpointing   + This option cannot be used with W-type files.   0  No processing option is required |

.

**EXAMPLES**

* The following example checkpoints the current sub-file.

dft\_fil \*file\_ptr;

dfckp(file\_ptr,0);

## THE DBCLR COMMAND MACRO :

* DBCLR lets you exit the program without closing any open sub-files and without generating a dump.
* Usage not recommended as it leaves sub-files in partially updated condition
* If you want to use exit and want to discard change, use DBCLS with ABORT option.

**FORMAT**

DBCLR

let’s you exit the program with DB file still open.

# TPFDF ADVANCED MACRO:

## DBCPY : COPY A SUB-FILE

The DBCPY command

* copies the sub file into pool files,
* close the original sub file and
* uses the pool copy for all subsequent TPFDF Commands on the sub-file ,
* Uses the default pool type for the file.

|  |  |
| --- | --- |
| **PARAMETERS** | **FUNCTION** |
| REF= and/ or FILE= | Specifies the file that we want to access |
| ALG or FADDR or ORD | Method used for identifying the sub-file that we need to access |
| TOADD | * Copies sub-file to specified prime block, where this field contains 4-byte file address of the target prime block. * New sub file will overwrite the data that was already there. * If you want to create a new pool sub-file, then either * Don’t specify this parameter OR * Specify file address of hex 0 for this parameter |
| HELD | Indicates that ECB is already holding the address specified by TOADD parameter |

**NOTES**

* The file addresses of the source and destination must have the same header and file address format types or an error is returned.
* If a new pool is allocated, it is created using the same header and file address format as the source
* This function sets up a 2-byte sequence SW00SEQ field in the SW00SR slot.
* This sequence number can be used to restore the same sub-file with DBRST command

**EXAMPLES:**

DBCPY REF=TRABSQ

DBCPY REF=TRABSQ, TOADD=(EBW000,HELD)

## dfcpy–COPY A SUB-FILE

This perform same function as DBCPY

**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);

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| file | File Pointer, Pointer to the base address of SW00SR slot of the file that we want to access |
| access | * It can be either of   + DFCPY\_ALG OR   + DFCPY\_FADDR OR   + DFCPY\_ORD * It indicate the method that should be used to access the sub file |
| acc | * It is the value to be used with access parameter to decide which sub file that we want to access |
| options | * DFCPY\_CREATE =   Create empty sub file by using pool blocks instead of copying original sub file   * DFCPY\_HELD =   ECB is already holding the FA specified by toa parameter   * 0 |
| toa | * Target Prime Block Address |

**RETURN CONDITIONS**

* Pointer to address of header of prime block of copied sub file

**Examples**

The following example copies a sub-file to pool blocks.

dft\_fil \*file\_ptr;

dft\_hdr \*block\_ptr;

block\_ptr = dfcpy (file\_ptr,0);

## DBRST: RESTORE A SUB-FILE

* The DBRST command restores a sub file previously copied using a DBCPY

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= And/ Or FILE= | * Specifies the file that we want to access |
| ALG or  FADDR or  ORD | * Specifies Algorithm or File Address or Ordinal, which is used to access particular sub file. * This detail corresponds to SOURCE SUB-FILE. |
| RSTRADD | * Target sub file where you want to restore the source sub file * If address is set to X’0’, then new sub-file is created and source file is restored to new sub-file. SW00FAD |
| HELD | * Restore a file that is held (DBOPENED with HOLD) |
| FLIP | * Interchange the contents of the source file and target sub file |
| FROMCHAIN | * restores the prime block of the source sub file to the prime block of the target sub-file. * Any overflow blocks in the source sub file are chained to the restored prime block of the target file. |
| SEQ = | * specifies an update sequence number. * The number you provide must match the sequence number contained in the sub-file that is specified by RSTRADD parameter * If numbers match, DBRST macro restores the sub-file and increase sequence number by 1. * Sequence number is placed in the prime block of the restored sub-file. * This parameter can be specified only if #BIT7 is set ON in the SW00OP1 field of the DBDEF for the file. |

* The prime block from the source sub file specified by the REF parameter is copied to the prime block for the target sub-file specified by the RSTRADD parameter.
* One of the following also occurs:

|  |  |  |
| --- | --- | --- |
| **FLIP** | **FROMCHAIN** | **ACTION OF OVERFLOW BLOCKS** |
| N | N | * Overflow Blocks from source sub-file are copied to blocks acquired from pool. * Overflow block from source sub-file and original overflow from target sub-file are released. * If Prime block of source is fixed file. NAB is reset |
| Y | N | * Prime block of target is copied to source. * Overflow block of source are chained to prime of target sub-file * Overflow block of target are chained to prime of source sub-file * No File overflow blocks are created or copied * No file blocks are released |
| N | Y | * Overflow blocks from source sub-file are chained to restored prime block of target file * Overflow block of target sub-file is released if prime block of source is pool file.. ELSE NAB is reset |
| Y | Y | * FROMCHAIN parameter is ignored. |

* The restored target sub-file (with any modifications you have made to the copy) becomes the sub-file that is currently open. You can continue processing it by using other z/TPFDF functions.
* When you use the SEQ parameter, the DBRST macro returns an error if

Number specified and the sequence number in the file specified by RSTRADD does not match.

In this case, Bit 6 is set in the SW00RTN field and the sub-file is not restored.

**EXAMPLES:**

DBOPN REF=TR00SQ,ALG=MI0ALG

DBRED REF=TR00SQ,REG=R4

MVC EBW000,SW00FAD

DBCPY REF=TR00SQ

MVC EBW004,SW00FAD

DBRED REF=TR00SQ,REK=R4,KEY1=(PKY=#TR00K80)

DBDEL REF=TR00SQ

DBCLS REF=TR00SQ,REUSE

DBRST REF=TR00SQ,FADDR=EBW004,RSTRADD=EBW000,ERROR=RSTERR

## dfrst–RESTORE A SUB-FILE

**FORMAT**

void dfrst(dft\_fil \*file, dft\_opt options, dft\_fad rstaddr);

void dfrst\_seq (dft\_fil \*file, dft\_opt options, dft\_fad8 \*rstaddr8, dft\_seq seq);

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| file | Pointer to the base address of SW00SR slot of file that we want to access |
| options | * DFRST\_FLIP   Flips sub-file specified by file and file having rstaddr as file address   * DFRST\_FROMCHAIN   Restores prime block of source sub-file to prime block of target sub-file. Overflow block of prime are now chained to restored prime block of target sub-file.   * DFRST\_HELD   Restore sub-file that is opened with DFOPN\_HOLD option   * DFRST\_SEQ   Restores the sub-file with correct sequence number.  Before restoring, check is made to make sure that sequence number provided matches with the sequence number of file whose address is rstaddr |
| rstaddr | * 4-byte file address of target sub-file to which we want to restore source sub-file * If address provided is 0, new sub-file is created for target sub-file |
| seq | * is an update sequence number. * The number you provide must match the sequence number contained in the sub-file 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 sub-file and increases the sequence number by 1. * This sequence number is placed in the prime block of the restored sub-file. |

**RETURN CONDITIONS**

The contents of the prime block from the source sub-file specified by the file parameter is copied to the prime block for the target sub-file specified by the file address in the *rstaddr* parameters

**EXAMPLES**

**Following example restores a sub-file**

The following example restores a sub-file that might be being held (by this or another application).

dft\_fad fad;

dft\_fil \*file\_ptr;

dfrst(file\_ptr,DFRST\_HELD,fad);

## DBSRT: SORT A FILE

* The DBSRT command creates a copy of a file and sorts it according to the KEYn parameters provided

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| INPUTREF= | Reference of the input sub-file whose LREC needs to be sorted into output file |
| REF= | Reference of the output sub-file |
| UP or  DOWN | This indicates whether LREC are organized in ascending (UP) or Descending (DOWN) order of key fields |
| KEYn or KEYLIST | Key parameters that should be used for sorting, along with their organization |
| RELEASE | Release SW00SR slot of the input sub-file after macro has completed processing |
| RELFC | * Release input file and release it from DASD * All overflow blocks are released. * If sub-file is indexed. * If algorithm was specified earlier, sub-file is de-indexed from specified path * if AUTOINDEX=YES, parameter is specified on DBDEF macro, input sub-file will be de-indexed from all indexes |

**HOW FILES LOOKS AFTER DBSRT:**

|  |  |
| --- | --- |
| **FILE** | **AFTER DBSRT** |
| INPUT FILE | Will remain unchanged |
| OUTPUT FILE | Will contain Sorted file with new LRECs (from input file) merged with old LREC (original content of file) |

**NOTES: -**

* Both files must be OPENED before calling DBSRT command
* Any keys that are active for the input files when DBSRT macro is called are used to select records from the input file
* The z/TPFDF product creates the output sub-file for you when there is no file address associated with the output file (for example, if the output file was opened without an ALG=, FADDR=, FADDR8=, or ORD= parameter) before sort processing.
* If the output file was opened in DETAC mode:
  + If the number of blocks in the output file exceeds the maximum sort batch file size (as defined with #TPFDBSB in the ACPDBE macro), the DBSRT macro will put the sorted output in a newly created pool file.
  + The output file originally associated with the SW00SR referenced by the REF or FILE parameter is emptied (any blocks previously chained to the prime block are released).
  + If the prime was a pool file address, it is also released.
  + Therefore, whenever the output file is in DETAC mode, if you need to reference the file address of the output file, you should reload the address from SW00FAD (or SW00FAD8).
  + Only the prime block will be in core after DBSRT processing has been completed.
* When DBSRT processing has been completed, the output sub-file remains open and must be closed by using the **dfcls** function before the ECB exits.
* If the RELEASE parameter value is not specified, the input file is closed internally without releasing the SW00SR.
* You can enter subsequent commands for the input file, but you must specify both the FADDR and KEYn (or KEYLIST) parameters on the next API for the input file.
* After sort processing has been completed, any new access of the input file will be with HOLD regardless of the parameter value specified originally on the DBOPN macro of the input file.
* The DBSRT macro does not change the input sub file.
* You cannot iSSUe additional z/TPFDF macros to the input file if the following conditions are true:
  + You specified the FULLFILE parameter.
  + The end-of-file indicator is set.
* If the AUTODEINDEX=YES parameter is specified on the DBDEF macro and an indexed sub-file is empty when the input sub-file is sorted, the input sub-file will be de-indexed from all its indexes and all corresponding pool file addresses will be released

**EXAMPLES:**

DBOPN REF=CR00SQ,FADDR=CRABCAR

DBRED REF=CR00SQ,REG=R4,BEGIN

DBIFB REF=CR00SQ,NEWREF=CR00SQA

DBOPN REF=CR00SQ,HOLD,DETAC

DBCRE REF=CR00SQ

DBSRT REF=CR00SQ,INPUTREF=CR00SQA,UP,RELEASE,

KEY1=(CR00KEY),

KEY2=(CR00NAM),

KEY3=(CR00CTY)

## dfsrt–SORT A SUB FILE

This function performs same function as DBSRT.

**FORMAT**

void dfsrt(dft\_fil \*file, dft\_fil \*input, dft\_opt options, dft\_kyl \*keylist);

void dfsrt\_pty (dft\_fil \*file, dft\_fil \*input, dft\_opt options, dft\_kyl \*keylist, dft\_pty pty);

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| file | Pointer to base address of SW00SR slot of output sub-file |
| Input | Pointer to base address of SW00SR slot of input sub-file |
| Key\_list | Address of sort key list of the output sub-file specifying the order into which we want to sort the LRECs |
| Options | * DFSRT\_FULLFILE   Sorts LRECs from entire input file. This parameter can’t be specified if input file is W-type   * DFSRT\_RELEASE   Releases SW00SR slot of the input   * DFSRT\_RELFC   Release input sub-file and delete it from DASD.   * 0 |
| pty | * Pool Type |

**Examples**

The following example selects LRECs from the sub-file in\_file\_ptr using key list structure selectkeys and puts them into sub-file out\_file\_ptr in the order described by the sortkeys key list structure

dft\_fil \*in\_file\_ptr;

dft\_fil \*out\_file\_ptr;

dft\_kyl selectkeys;

/\* 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\_nbrkeys(&selectkeys, 1);

df\_setkey(&selectkeys, 1, offsetof(struct gr95sr, gr95key),

1, DF\_EQ, &pky, 0, DF\_UPORG, DF\_CHAR);

dfkey(in\_file\_ptr, &selectkeys);

/\* sort the sub-file 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\_file\_ptr, DFSRT\_RELEASE, &sortkeys);

## DBMRG: MERGE TWO FILES

* The DBMRG allows to merge two TPFDF files or sub files into one output file

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| INPUTREF= | Specify name of the file to be submerged |
| REF= | Ref name of sub-file where merged records will be stored |
| UP or DOWN | Specifies that the LRECs are organized in Ascending (UP) or descending (DOWN) order |
| FULLFILE | Merges LRECs from entire input file to the output sub-file |
| RELEASE | Releases the SW00SR slot for the input sub-file after MACRO has completed processing |
| RELFC | Release input sub-file and delete it from DASD.  If sub-file is indexed  If algorithm is specified earlier, then sub-file is de-indexed from specified path  If AUTODEINDEX=Y is specified in DBDEF, input file will be de-indexed from all of its indexes |
| KEYn | Specifies the key parameter to be used with this macro |

**NOTES:-**

* Both References in REF= and INPUTREF= should have been opened previously
* One cannot have the output file opened in the DETAC mode.
* Should ensure input sub-file and target sub-file are in the same-sorted order. IF they are not originally use DBSRT before merging
* Any keys that are active for the input file when you call this macro will be ignored, but are preserved. Use the DBSRT macro if you want to select records from the input file.
* You cannot issue additional z/TPFDF macros to the input file if the following conditions are true:
  + You specified the FULLFILE parameter.
  + The end-of-file indicator is set.
* However, you can issue the DBCLS macro with the REUSE parameter specified and then continue with additional z/TPFDF macros

**EXAMPLE**

DBMRG REF=GR26DF,INPUTREF=GR25DF, \*

KEY1=(R=GR25KEY,L==AL2(L'GR25KEY))

## dfmrg–MERGE LOGICAL RECORDS FROM TWO SUB-FILES

**FORMAT**

void dfmrg (dft\_fil \*file, dft\_fil \*input, dft\_opt options, dft\_kyl \*key\_list);

|  |  |
| --- | --- |
| PARAMETER | FUNCTION |
| file | Pointer to base address of the SW00SR slot of the file that we want to access.  It is the sub-file in which merged file will be stored |
| input | Pointer to the SW00SR slot of the input sub-file that will be merged into the sub-file reference by ***file*** |
| option | Processing options for this function. Use following   * DFMRG\_FULLFILE   merges LRECs from the entire input file to output file referenced by *b*   * DFMRG\_RELEASE   Releases SW00SR slot of the input file after processing merge   * DFMRG\_RELFC * Release input sub-file and delete it from DASD.   + If file is pool file   + Prime block is also released   + If al * 0 |
| key\_list | Pointer to the key list of the merged output file |

**EXAMPLES**

The following example merges the two open sub-files referenced by file\_ptr and input\_ptr. All the LRECs go into the file\_ptr sub-file.

/\* 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);

# TPFDF INDEXING SUPPORT

* Algorithms and Keys provide the identity for an LREC in a particular sub-file,
* But to locate this LREC, TPFDF has to search all LRECs in the sub-file sequentially trying to match the key specifications. This slows down the search considerably when the sub-file has a number of LRECS and hence a number of overflow blocks and I/O's
* To reduce the time taken for the search, indexing is used.
* TPFDF Supports organizing the database into an indexed structure hence enabling to construct a Hierarchical database
* Indexing reduces amount of physical I/O processing required, makes accessing the LREC faster.
* TPFDF Supports 3 Mechanisms to Implement Indexing
* Block Indexing
* B+Tree Indexing
* TPFDF Indexing

The 1st and the 2nd Indexing mechanisms are used for quick location of LRECs in a sub-file

The 3rd Index mechanism helps in organizing the database into a Hierarchical structure.



## BLOCK INDEXING

* For Files using the Block Indexing support TPFDF Maintains TLRECs (Technical LREC's) in the prime block of the sub-file
* There exists one TLREC for each overflow block,
* Each TLREC contains the File address of the overflow block and the Key field of the first LREC in the overflow block on which the file is Organized

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Size | 02 | Addr | rcc | Identifying data |

**EXAMPLE OF INDEX TLREC**.

* In this, each TLREC contains the data extracted from 1st LREC of the overflow block.
* z/TPFDF compares each TLREC with the LREC key to be searched. Once it find that LREC key is lesser than that in TLREC, TPFDF search in previous block for first instance of that.
* TLREC's have a primary key 02,
* To implement block index support, set &SW00SKE Field in the DSECT has to be updated with the key field length in the LRECs on which the Block indices have to be set up
* The TLRECs are held only in the Prime block and that poses a limit on the number of overflows that can be block indexed, The remaining LRECs are searched sequentially

**PROCESS OF GENERATION OF TLRECs**

* TLRECs are not immediately generated if file expands and overflow blocks are needed.
* During pack of the sub-file, z/TPFDF product determines how many chains are there and stores that in the first TLREC.
* During next pack, z/TPFDF product generates that many TLRECs, initialize them and determine new number of chains.

## B+TREE Indexing

* Similar to Block indexing support, it uses index TLRECs to identify the first data LREC in each block of a sub-file
* B+ tree file consists of a data file, which contains LRECs and an index file, which contains TLRECs. B+ Tree Index File (called nodes) consists of blocks internally maintained by z/TPFDF product. Index file has its own file-ID, DSECT and DBDEF statements.
* Prime block of B+Tree Index File (root node) is pointed to by the header in the prime block of B+ tree data file.
* B+ tree has following advantages.
  + Dynamically updates TLRECs
  + Has unlimited number of TLRECs
  + Can use mixed key organization for file operation
* TLRECs not maintained in Prime block of sub-file , are maintained in separate B+tree structure
* TPFDF retrieves the block containing the LREC directly after looking at the B+tree index , The LRECS are searched sequentially in the block later
* Each sub-file using B+tree indexing has a B+tree structure for it

**B+ TREE DATA FILE CHARACTERSTICS**

* To use B+Tree indexing, set the DBDEF macro parameters, equivalent DSECT parameters, or equivalent default values for a B+Tree data file as follows:
* SW00OP1 bits 2, 3, and 6 must be OFF.
* SW00OP3 bit 5 must be ON.
* FARF6 is not supported for B+Tree data files.
* In the DBDEF macro,
  + the F6PRP and F6OFP parameters (OP5 bits 3 and 4 must be off) must be set to NO (the default) for B+Tree data files.
* The SW00RBV algorithm value cannot be #TPFDB0D.
* The SW00TQK value must be greater than 4.
* If present, the PIN value must be less than or equal to 50.
* The TYP value must be R.
* The NOC variable cannot be present.
* The SKE variable cannot be present.
* The NLR variable cannot be present.
* Also, the B+Tree data file DBDEF must have the following:
* A PKY statement to define default keys
* NODEID=fileid (where fileid is the &SW00WID value shown in the associated B+Tree index file)
* KEYCHECK=YES
* UNIQUE=YES
* DELEMPTY=YES if the DBDEF includes statements for recoup to perform multiple ECB chain chasing (see [Multiple ECB chain chasing](http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.7/com.ibm.ztpf-ztpfdf.doc_put.07/bdfd1/mecb.html?lang=en-us&cp=SSB23S_1.1.0.7#mecb) for information about multiple ECB chain chasing).SW00OP3 bit 5 is set ON to indicate Block indexing for a TPFDF File
* Removes restrictions posed by Block indexing on the number of blocks that can be indexed , Poses a host of other limitations on TPFDF Command macros and Option byte settings
* B+tree indexing support has been included to the TPFDF Program product from PUT level 5 onwards

## TPFDF STANDARD FILE INDEXING

* TPFDF File indexing allows organizing the database into a hierarchical structure,
* The structure is made up of parent files called as Index files and child files called as detail files
* LRECs in the index file are called as the Index *LRECs* and typically each index LREC would have a pointer to a detail file or to a subsequent index file (2nd Level Index)
* Several Index files can refer to the same detail file , This is called as Multiple indexing
* One Index LREC can be pointing to more than one indexed detail file
* The Top level index files should be fixed files ,
* The subsequent indices and the detail files are usually pool files with algorithm #TPFDFFF
* Index LRECs apart from containing the file address of the detail file also have Key fields that relate to the detail file
* The detail files can be directly opened using an algorithm string ,
* The ALG string in this case is made up of at least two parts where one part of the ALG string is for deciding the sub-file of the top level index and the other part is for searching the Index file LRECs as the key field
* The Index files are maintained by TPFDF , using indexing allows for easy access of detail files and reduces the number of TPFDF Command macros in application programs
* TPFDF Provides 2 command macros to create indices and to remove indices for detail files
* DSECT and the DBDEF have to be defined for an index structure specially.

## DBIDX: CREATE AN INDEX

* The DBIDX command macro allows you to create one or more index reference to a sub-file.
* We can index one or more paths

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= or FILE= | This specifies the file that we want to access |
| ALG= | Algorithm argument used to identify the sub-file that we want to access.  If the sub-file you are accessing is contained in a detail file or intermediate index file defined with the #TPFDBFF algorithm, the z/TPFDF product uses the algorithm argument to locate the sub-file |
| PATH | * It specifies the path for the detail sub-file using index support. * If there is only 1 index path, don’t specify this parameter * IT can be either   + Path number= 2 byte path number.   + All = Specifies all paths. |

Optionally using the DBADD with INDEX an LREC to the detail file automatically updates the index for this detail file

**EXAMPLES**:

DBOPN REF=RRF2BR,HOLD,SPACE=#RRF2BRWKA

DBCRE FILE=RRF2BR,ERROR=DBCRERR

DBIDX REF=RRF2BR,ALG=MI0NAM

## dfidx–Create an index reference

**FORMAT**

void dfifx\_alg (dft\_fil \*file, dft\_opt options, dft\_alg \*alg);

void dfifx\_alg\_pth (dft\_fil \*file, dft\_opt options, dft\_alg \*alg, dft\_pth pth)

|  |  |
| --- | --- |
| **PARAMETERS** | **FUNCTION** |
| file | Pointer to the base address of SW00SR slot of the file that we want to access |
| alg | Pointer to an algorithm argument that identified the sub-file |
| options | Processing options for this function   * DFIDX\_PATH\_ALL   Indexs all the paths. Do not use this parameter if using pth parameter   * 0 |
| pth | Path number for a detail sub-file using index support  Value is defined in DBDEF macro and is decimal number |

* There must be a detail file available
* Ensure that relationship of the index files to the detail file has been defined with DBDEF macro by DBA
* Path 0 is default path.
* No actual index structure needs to exist before you index the sub-file. 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 z/TPFDF product creates the required index structure automatically when you call the ***dfidx*** function.
* If you index a sub-file 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 sub-file (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);

## DBDIX : DELETE AN INDEX

The DBDIX command macro allows you to delete index or indices for a indexed detail record

|  |  |
| --- | --- |
| **PARAMETER** | **FUNCTION** |
| REF= or  FILE = | Specifies the file that you want to access |
| ALG= | Identifies the sub-file that we want to access |
| PATH= | Specifies the path for a detail sub-file using index support.  If there is only one index path, do not specify this parameter  It can be either of following   * Pathnum   Path number or label of 2-byte field that contains the path number. Number of index paths used is defined by your DBA   * All   Specifies all the paths |

**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

**PROGRAMMING CONSIDERATION**

* There is no INDEX option with DBDEL so on deletion of the detail sub-file the application has to delete the indices using the DBDIX command
* Using the DBDIX macro can result in having a detail file with no corresponding index file pointing to it.
* If an intermediate index becomes empty, the z/TPFDF product releases the block and deletes any references to the subfile in higher-level indexes

**EXAMPLES:**

DBOPN REF=RRF2BR,HOLD,ALG=MI0NAM

DBDEL FILE=RRF2BR,ALL

DBDIX REF=RRF2BR,ALG=MI0NAM

DBCLS REF=RRF2BR,RELEASE,RELFC

## dfdix–DELETE INDEX REFERENCES TO A SUB-FILE

**FORMAT**

void dfdix\_alg (dft\_fil \*file, dft\_opt options, dft\_alg \*alg);

void dfdix\_alg\_path (dft\_fil \*file, dft\_opt options, dft\_alg \*alg, dft\_alg pth);

|  |  |
| --- | --- |
| PARAMETER | FUNCTION |
| alg | Pointer to the algorithm argument that identifies the sub-file |
| file | Pointer to the base address of SW00SR slot of the file that you want to access |
| options | Processing Options   * DFDIX\_PATH\_ALL * 0 |
| pth | * Path number for detail sub-file using index support. * Value is defined in DBDEF macro and is decimal number. * Default path number is 0 |

**EXAMPLES**

The following example deletes an index reference to a sub-file (path 1 only).

dft\_fil \*file\_ptr;

char member\_number[10];

dfdix\_alg\_path(file\_ptr,0,member\_number,1);

## TPFDF PARTITIONING AND INTERLEAVING

* The ordinals in a File can be divided into different parts such that each of the group of ordinals (sub-files) belong to a different partitions or interleaves
* This is a logical mechanism of separating the database for different purposes
* The number of partitions or interleaves are specified in the DSECT in the SW00PTN and SW00ILV parameters respectively , cannot be implemented using Miscellaneous files

**EXAMPLE :**

If there are 100 ordinals for a TPFDF file and if it has 4 partitions , then partitions 1 thru 4 each would have 25 ordinals , 0..24,25..49,50..74,75..99

Ordinal 0 for partition 1 is 0 , while for partition 2 it is 25 , 50 for partition 3 and 75 for partition 4

If there are 100 ordinals for a TPFDF file and if it has 4 interleaves, then interleaves 1 thru 4 each would have 25 ordinals , Interleave 1 would consist of ordinals 0,4,8,12,16,20… 96 . Interleave 2 would consist of ordinals 1,5,9,13,17,21… 97 . Interleave 3 would contain ordinals 2,6,10,14,18,22… 98 and interleave 4 3,7,11,15,19,23…99

Ordinal 0 for interleave 1 is 0, While for interleave 2 it is 1 , 3 for interleave 3 and 4 for the last interleave

* Advantage of partitioning the database is the ease of adding a new partition
* Disadvantage of partitioning comes when one needs to increase the number of ordinals in a partition
* Advantage of Interleaving is that the number of ordinals in an interleave can be increased easily
* Disadvantage is the number of interleaves cannot be increased
* Two parameters are provided with TPFDF that can be used to specify the partition number and the interleave number , These are allowed with most of the TPFDF Command macros

**The INTERLV= Parameter**

* Specify a Register an absolute value or a label indicating the location of a 2-byte field containing the number of interleave that you want the command macro to perform operation on , Can be used with DBOPN,DBRED,DBADD,DBDEL

**The PARTITN= Parameter**

Specify a Register an absolute value or a label indicating the location of a 2-byte field containing the partition number that you want the command macro to perform operation on,

* Can be used with DBOPN,DBRED,DBADD,DBDEL

## USING UNIQUE KEY

* Unique keys can be used to logically relate LRECs with different primary keys holding related data
* Instead a unique key field that relates LRECs can be defined as part of each key
* TPFDF generates 4 unique values when requested by application programs using the DBUKY command macro
* Before adding a new passenger to the file in the example above, The application should request for a unique key and use this value as part of the unique key field of each of the common keys 80, 90 and C0.
* TPFDF ensures that the value generated is unique for every new request
* An additional 18 bytes needs to be allocated as part of the TPFDF header (10 spare bytes, 4 bytes STDFUK & 4 bytes STDEUK)
* The unique key generates value from 00000001 to 7FFFFFFF after which the value is recycled
* Each sub-file has its own unique key field
* A DB0126 dump is issued to indicate that the unique key value has exceeded the maximum value
* Setting #BIT 4 of SW00OP1 on indicates a checkpoint to be taken after each DBUKY command macro
* For Example,

If a TPFDF sub-file holds Passenger names in Key 80,

Address of the passengers in Key 90 and

Meal Preference in Key C0,

A common key would be required to relate all the 3 LRECS.

Repeating the name field in Keys 90 & C0 is an option,

The dis-advantage being redundancy of the name field and wastage of LREC space

## ADD CURRENT FILES

* ADD CURRENT files are TPFDF files which can be defined to only hold certain number of records and then wrap around, thereby retaining only the most recent data
* Ideally suited for log files (Macro trace table is a wraparound log table)
* Do not have to write maintenance programs to delete old data in the log file
* The maximum amount of data to be held can be controlled using the SW00NOC (0-255) parameter of the DSECT
* SW00OP1, #BIT 2 should be set on to indicate an add-current file (Do not set this bit ON for normal files, Data could be lost)
* The oldest LREC is overwritten after the maximum number of chains has been reached adding LRECS
* No specific settings required in the DBDEF
* No special coding consideration to be taken while writing applications using ADD CURRENT files

## PUSH DOWN CHAINING

* TPFDF provides a mechanism called **pushdown chaining** for applications requiring fast access to the most recently added records
* Using this feature of TPFDF, All addition of logical records are only made to the prime block of the sub-file
* If there isn't space in the prime block after the add, The LRECs from the prime block are moved to a new overflow block, The prime block initialized and the new LREC added to the prime block
* SW00OP1, #BIT 3 has to be turned ON to implement push down chaining
* Push down chaining involves very little data movement on add's at the same time DBRED's for recent LRECS are very fast as not too many chains are read
* DBSRT command cannot be used with push down chained files

# TPFDF MACRO DESIGN

**Contents**

* Design Introduction
* TPFDF Physical Design
* Indexing Options
* DBDEF
* DBDEF Data Structures
* DBDEF Organization
* DBDEF Structure
* Coding DSECT
* Initial Area & Global Set Symbols
* Meaning of the Global Indicators
* TPFDF Block Header
* LREC definition
* User Fields
* Algorithm
* Default Keys
* Indexing Parameters
* Database Optimization
* Block Sizes
* Tips on using Indexing

## PHYSICAL DESIGN – GUIDELINES

The following are good practices that should be adapted while physical design is under construction.

**WORKLOAD ESTIMATION**

* Though precise analysis of the workload is often impossible, a rough estimate will be enough.
* The transactions and the transaction behavior is very much essential to calculate, the primary transactions, their required response time; to determine the expected frequency of such transactions; work load that will be due to this transaction.
* The access patterns helps us in determining the following
* Single record access
* Access to a group of records
* Sequential access to every record of a type.
* Frequency and pattern of inserts/deletes and updates.
* A data characteristic on the other hand, helps us understanding the approximate number and size of records in the database.

**ACCESS TIME ESTIMATION**

* For majority of database systems performance is dominated by processing cost.
* A good physical design ensures that there is less access time involved.

## CALCULATING LOGICAL I/O

* Calculating logical I/O estimates are useful for building up alternative models of workload with several important transactions.
* The following access pattern’s characteristics should be measured.
  + Single record retrieval,
  + Range Retrieval,
  + Whole table access.
* A good database design requires all these steps to be taken enough care.

|  |  |
| --- | --- |
| Physical Database organization | * All the logical records should be put into physical files. * Relate physical files to each other through indices and imbedded addresses. * Relate logical schemas that will be used together and put them in the same physical file. * Ordinal range / partitions/ interleaves should be taken care. |
| Block Size | * Choose the correct block size for both prime and overflow blocks. * Performance has a direct correlation with the block size. * Alternate block size can be a good option. |
| Record Organization | * Ensure the primary keys are properly defined. * The primary keys should have enough room for insertions in between. * Ensure that the primary key (for LREC- singular/composite) is properly identified. * Ensure that the file organization is defined correctly. |
| Access Method | * Algorithm, Indexing, Block Indexing, B+ trees * In case of indexed file, ensure that paths are defined appropriately. * In case of block indexing, highest technical record and key extract size should be checked. |
| Pools | * Ensure that the primary and alternate pool types for fixed overflow and prime pool files are defined correctly. |
| Others / Soft options | * Ensure that the DSECT name, file identifiers, logical record types/names and field names are right. * Are the option bits set as desired? |

## INDEXING

**ACCESS TIME ESTIMATION**

* The detail files can be directly opened using an algorithm string, The ALG string in this case is made up of at least two parts where one part of the ALG string is for deciding the sub-file of the top level index and the other part is for searching the Index file LRECs as the key field
* The Index files are maintained by TPFDF, using indexing allows for easy access of detail files and reduces the number of TPFDF Command macros in application programs.
* Various flavors of basic TPFDF indexing can be designed
* Single record access
* Simple indexing
* Multiple indices pointing to single detail file
* Multi-Level Indexing
* Single index to multiple detail files
* Any index LREC (LREC that holds the address of the detail file) using TPFDF basic indexing should be in the following format

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SIZE | KEY | ADDR | RCC | KEY | DATA |

* Where size and keys are 3 bytes long, followed by a 4-byte address of the detail file and a 1 byte RCC. The ADDR and the RCC pair are repeatable.
* The key data following the RCC should have "key fields" organized UP or DOWN.

## CODING DSECT

* A DSECT macro definition must be created to describe each file that will be access from an application program.
* These macros should be coded as every macro is coded in the TPF environment.
* Once these macros are coded, they should be placed in the macro library for applications and TPFDF utility programs to access.
* The process of coding a DSECT is sliced up into functional units for better understanding.
* The following are the areas that will be addressed.
* Initial Area & Global Set Symbols.
* Block Header
* LREC definition (Single/Multiple)
* User Defined Fields in the LRECs
* Algorithm related areas

## INITIAL AREA & GLOBAL SET SYMBOLS

**SAMPLE MACRO:**

MACRO LIBR NAME = LR0EZZ01

&LABEL LR0EZZ &REG=,&SUFFIX=,&ORG=,&ACPDB=

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* TITLE Incoming Host-to-host communication table (detail) \*

\* CREATED 30-Dec-1988 \*

\* \*

\* \*

\* CHANCE HISTORY DATE DESCRIPTION \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GBLB &LROEZZ1 1ST TIME CALLED SWITCH

COPY DBGBL COPY TPFDF GLOBAL DEFINITIONS

COPY DBLCL COPY TPFDF LOCAL DEFINITIONS

&NAM SETC 'IHLT' DOC NAME

&DATE SETC '30DEC88' UPDATE DATE

&VERS SETC '01' VERSION NUMBER

&SW00WID SETC '0609' FILE ID

&SW00WRS SETC 'L1' BLOCK SIZE

&SW00ARS SETC 'L4’ ALTERNATE BLOCK SIZE

&SW02FIL SETC 'LR0EZZ' FILE DSECT NAME

&SW00OP1 SETC '00000000' OPT BYTE1

&SW00OP2 SETC '00000110' OPT BYTE2

&SW00OP3 SETC '00000000' OPT BYTE3

&SW00TQK SETC '15' HIGHEST TLREC

* Setup the DSECT name according to the convention.
* Setup values for name, date & version fields.
* Setup values for global fields.
* A good practice is to not set values for those fields that are not required.
  + For example, if you are not using partitions then don’t set values for SW00PTN.
* The values that are coded in the DBDEF will override the values set here in the DSECT.
  + In such cases, don’t code the value in the DSECT.

**MEANING OF THE GLOBAL INDICATORS**

|  |  |  |
| --- | --- | --- |
| **FIELD** | **NEEDED?** | * **FUNCTION** |
| SW00WID | Yes | * Specifies the file ID, which is a 2-byte value in the standard header of every block. * Each DSECT should have a unique ID. |
| SW00WRS | Yes | * Specifies the size of the prime block used in the file. |
| SW00ARS | No | * Specifies the size of the overflow blocks to be used in the file. * If not assigned the size is taken to be the same as WRS. |
| SW00BOR | Yes(F) | * Needed only for fixed and miscellaneous files. * It is used to specify the beginning ordinal for the file. * For a fixed file, always set the base ordinal number to zero. * If this is a miscellaneous file, specify the symbol (usually defined in SYSEQ) that defines where the miscellaneous file starts in the miscellaneous fixed file type. |
| SW00EOR | Yes(F) | * Needed only for fixed and miscellaneous files. * It is used to specify the ending ordinal for the file. * For a fixed file, non-partitioned, always set the base ordinal number to -1. * If this is a miscellaneous file, specify the symbol (usually defined in SYSEQ) that defines where the miscellaneous file starts in the miscellaneous fixed file type. * For a partitioned file, specify the number of prime blocks in each partition, not the total number of prime blocks in all partitions. |
| SW00ILV | No | * The number of interleaves. * Defaults to zero, meaning no interleaves. |
| SW00PTN | No | * The number of partitions that are there. * Defaults to zero, meaning no partitions. |
| SW01E0# | Yes(F) | * &SW01EO# specifies the FACE-type end ordinal number. * For non-partitioned & non-interleaved, set value to –1. * For partitioned or interleaved, set it to the total number of prime blocks in the file. |
| SW02FIL | Yes | * Set the DSECT macro name. |
| SW00NLR | No | * Set only when file used algorithm #TPFDB0D. * Specifies the number of fixed-length LRECs in the record. |
| SW00NOC | No | * Specifies the number of blocks to use in implanting the current file. * Assign only when OP1 bit 2 is set; this limits the number of overflow blocks added as overflows. |
| SW00RBV | Yes | * Defines the algorithm used in this file. * For detail pool file, set it to #TPFDBFF. |
| SW00RCT | Yes(F) | * Specifies the file’s record type as defined in the Face Table. |
| SW00PIN | No | * Specifies a 0-100 value which will be used as the packing threshold. |
| SW00REF | Yes(T) | * This is valid only for T-type files. * Specifies the reference name of the W-type file that contains the T-type File. * Defaults to GW01SR. |
| SW00SKE | No | * Specifies that Block Indexing Support is used. * Set this equal to the length in bytes, of the key fields to be removed from the first LREC of the overflow blocks for TLRECs. |
| SW00TQK | No | * Specifies the highest technical LREC. * For block indexing should be more than 2. * For B+ support should be more than 4. |
| SW00TYP | No | * Determines the type of the file. * If omitted taken from the 2-char of the file name. |

**BLOCK HEADER**

* There are two things to be coded in the block header.
  + The TPF header area and
  + The TPFDF header area.
* If extra information needs to be put in the prime header area that area should be allocated after the standard area.

**EXAMPLE:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* STANDARD TPFDF HEADER \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LR0EHDR&CG1 DS CL16 STANDARD FILE HEADER

DS CL10 STANDARD TPFDF HEADER

LR0EVAR&CG1 EQU \* START OF VARIABLE USER-AREA

\* The user header area can be placed here \*

LR0EHDL&CG1 EQU LR0EVAR&CG1-LR0EHDR&CG1 HEADER-LENGTH UP TO LR0EVAR

**LREC DEFINITION**

* There are three important fields that each LREC is associated with.
  + 1. **REC =**

Indicate LREC size if LREC is fixed length LREC Or

= 1 if it’s a variable sized LREC

* + 1. **SIZ =**

Size of the variable length LREC

It includes 3 bytes for Size field and Key

* + 1. **KEY. =**

LREC ID,

It is used to differentiate between different type of LRECs.

**EXAMPLE**

ORG LR0EHDR&CG1

LR0EREC&CG1 DS 0CL1 1ST RECORD START (1=VARIABLE,ELSE SIZE)

LR0ESIZ&CG1 DS H SIZE OF LOGICAL RECORD

LR0EKEY&CG1 DS X LOGICAL RECORD IDENTIFIER

AIF ('&LR0ESR1' EQ '1').KEYEQ GO IF NOT FIRST ISSUE

**USER FIELDS**

* Every LREC in a file contains a LREC ID, or primary key, that is used for identification.
* More than one LREC can contain the same LREC ID value.
* Different LREC IDs are used in a file to differentiate between the types of LRECs in the file.
* The example shows how to define keys.
* Don’t use LRECS X’01’ to ‘0F’ and X’F0’ to ‘FF’. IBM reserves them.
* X’00’ cannot be used.
* The user fields should be holding the rest of the values.
* Define primary keys with enough ‘gap; between two primary keys to mark insertions easy.
* Other user-defined fields are used to define the fields of the LREC.

**EXAMPLE**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EQUATE OF LOGICAL RECORD KEYS (KEY AND LENGTH) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#LR0EK80 EQU X'80' LOGICAL RECORD KEY X'80'

#LR0EK90 EQU X'90' LOGICAL RECORD KEY X'90'

LR0EORG&CG1 EQU \* START OF LOGICAL RECORD DESCRIPTION \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION OF F I R S T LOGICAL RECORD TYPE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LR0EA80&CG1 DS 0CL30 KEYAREA

LR0ENUM&CG1 DS CL10 member number

LR0ENAM&CG1 DS CL20 surname

LR0EE80&CG1 EQU \* END OF LOGICAL RECORD WITH KEY = X'80'

## ALGORITHM

* For the entire detailed and intermediate-index file, algorithm should be defined.
* Algorithms need not be defined for top-level index and files that are not part of a basic index structure.
* The size of the algorithm string is calculated using the DSECT definition of the details or index-index file.
* The labels xxxx@nBEG and xxxx@nEND are used to fence the algorithm area.
* xxxx-is the first four byte of the application name and n is the path number.
* &SW00RBV set to '#TPFDBFF' for an indexed file.
* Bit 3 of &SW00OP3 set to 1 for an indexed fixed file or 0 for an indexed pool file.

**EXAMPLE**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ALGORITHM DESCRIPTION \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ORG LR0EREC&CG1

LR0E@0BEG&CG1 EQU \* PATH 0 DESCRIPTION

LR0E@0...&CG1 DS

LR0E@0...&CG1 DS

LR0E@0...&CG1 DS

LR0E@0END&CG1 EQU \*

DEFAULT KEYS

* Default Keys are used in TPFDF to maintain the organization of the LRECs when added to the sub-file.
* Applications can also use these keys while the records are read from the sub-file.
* Default keys are optional but for B+ tree indexing; B+ tree uses these keys to index data.
* Default keys, if used, should be defined for all the LREC types in the file.
* If the definition is incomplete, the system hits an error and exits.
* Default key goes with organization; meaning we need to have the LRECs UP or DOWN organized to use default keys.
* This feature cannot be used for P-type files, T- type files, Add Current Files, Index files, Pushdown Chaining files.

## INDEXING PARAMETERS

* The index file’s DSECT should contain definition of the fields in the index LRECs.
* Global symbols that specify how the index file is contained in physical records, including &SW00RBV to specify the algorithm used to access the top level of the index file.
* The DSECT of the Indexed file should contain &SW00RBV set to '#TPFDBFF' for an indexed file.
* Bit 3 of &SW00OP3 set to 1 for an indexed fixed file or 0 for an indexed pool file.
* The statements that define the fields in the ALG string associated with each indexing path.

## GUIDELINES – LOGICAL RECORDS

* Variable length records are preferred for data independence and flexibility.
* Use fixed length records only for simple files.
* Organize the logical records in the order of their physical keys.
* If logical records are related to one another; used together always, organize them so that each instances of this set is stored together.
* Use organization whenever possible to optimize searches and updates.
* Store repeating groups at the end of the LREC and precede them with the count.
* It’s easy to organize the sequence key immediately after the Primary key.
* Organize the record so that the secondary keys come after the sequence keys with descending priority.
* Use Primary key irrespective of the number of LREC items present.

## BLOCK SIZES

* Optimum selection of block sizes and the choosiest algorithm should keep the number of chains in any sub-file to be less than 3-6 chains.
* There are two trade-off criteria when it comes to choosing a block size.
* If the block is larger, there may be lot of wastages.
* If the block is smaller, there will a lot of I/O involved.
* The challenge is to strike a balance.
* A few thumb rules for block selection

FOR UNIFORM DISTRIBUTED SUB FILES – STATIC GROWTH:

|  |  |
| --- | --- |
| **FILE SIZE (EXCLUDING TPF & DF HEADER, UKY AREA & TRAILER)** | **OPTION** |
| 1 byte to 318 bytes | 381 block |
| 319 bytes to 993 bytes | 1K block |
| 994 bytes to 4033 bytes | 4k block |

FOR UNIFORM DISTRIBUTED SUB FILES – DYNAMIC:

|  |  |
| --- | --- |
| **FILE SIZE (EXCLUDING TPF & DF HEADER, UKY AREA & TRAILER)** | **OPTION** |
| Max 650 bytes | 381 P + 381 O |
| 650 bytes to 1300 byte | 1K P + 381 O |
| 1300 bytes to 5100 bytes | 1k P + 4K O |
| 5100 byte and more | 4K P + 4K O |

* For irregular distributed sub-files
  + most of the sub-files within the file require a space less than standard block.
  + but a few require a larger block.
  + Use the block size that fits most of them as the prime and use a larger block as the overflow.
  + Most sub-files require slightly large block than the standard block.
  + Some require multiples of the standard blocks.
* Use the standard size prime block and dynamic size overflow block.

## TIPS ON USING INDEXING

* Create Index operations may fail owing to the fact that some other ECB could have created the index in between the operation to read the index and the create operation .Use hold options in such case.
* HOLD is quite a huge performance hitter and should be used with caution.
* The other performance hitter is the checkpoint option used with the index files.
* SW00OP3 bit 4 indicates that the updates made to the index should be reflected on the DASD immediately, making this a performance degrader.
* But, this option still can’t be avoided as opening index with HOLD forces the index file to be in core till the detailed file is close, leaving the DASD not in sync with the core value.

# TPFDF UTILITIES

Following terms will be discussed in this section

* Capture Restore Utility Information & Statistics Environment (CRUISE)
* TPFDF Online Maintenance – ZUDFM Commands
* OAI (ACCESS)
* OAI/DBTAB (DEF)
* OAINIT (INIT)
* OA (FAD)
* OA\*(DISPLAY)
* OAA (ADD)
* OAX (DELETE)
* OAR (REPLACE)
* OAP (PACK)
* OAL (LOG)
* OAE (RESTRICT)
* OAZ (TOD)



## CRUISE – INTRODUCTION

* CRUISE is a validation, capture and recovery tool for TPF database administrators.
* CRUISE is a process of recovering and validating data, controlling pool space, and creating statistics.
* It is based on the file information that is stored in the DBDEF.
* It is a versatile and easy method of managing the TPF database.
* The information that is required by the CRUISE is contained in a parameter table.
* The parameter table contains the default values and prepares for the application use.
* It is created, set up, named and maintained by the database administrator.
* This simplifies the TPF database maintenance efforts.
* Different parameter tables can run on different loosely coupled processors.
* This shortens the processing time for CRUISE functions.

## TPFDF ONLINE MAINTENANCE – ZUDFM COMMANDS

**TPFDF ONLINE MAINTENANCE – ZUDFM COMMANDS**

* OAI – File access using ID or DSECT name
* OAI/DBTAB – Display DBDEF table items
* OAI/DBTAB – Display DBDEF table items
* OAINIT– Initialize files
* OA – Display the contents of a sub-file
* OA\* – Display LRECs in a sub-file
* OAA – Add LRECs to a sub-file
* OAX – Delete LRECs from a sub-file
* OAF – Display Forward & Backward chaining
* OAP – Pack a sub-file
* OAL – Display recent OA commands
* OAE – Restrict Table Handling
* OAZ – Convert time

## OAI(ACCESS) – FILE ACCESS USING ID OR DSECT NAME

**FUNCTION:**

* + Links to a sub-file when the ID or DSECT name is known.
  + Displays the information about the linked file.
  + Links to the first sub-file, if no parameters are given.

**FORMAT:**

|  |  |
| --- | --- |
| OAI/FILE or  ACC FILE | Display info about file with specified filename.  ORD 0 IS SET UP AS FAD FOR FURTHER OA-ENTRIES |
| OAI/ID | Display info about file with specified ID  ORD 0 is set up as fad for further OA entries |
| OAI/ID/CREATE | Create an empty pool file with specified file-header |
| OAI/ID/ALG-XX#CC#XXX | Calculate fixed ordinal with given algorithm |
| OAI/ID/ORDC | Set up ordinal in dec/ for specified file |
| OAI/ID/ORDX.... | Set up ordinal in hex/ for specified file |
| OAI/ID/PARTITN....  OAI/ID/PTN…. | Calculate ord-0 of specified partition and  Also set up partition for following entries |
| OAI/ID/INTERLV....  OAI/ID/ILV.... | Calculate ord-0 of specified interleave and  set up interleave for following entries |
| OAI/ID/ALG-.../PARTITN..  OAI/ID/ALG-.../PTN.. | Calculate ord by alg in specified partition |
| OAI/ID/ORDC.../PARTITN..  OAI/ID/ORDC.../PTN.. | Set up spec. ORD in specified partition |
| OAI/ID/ALG-../INTERLV.. OAI/ID/ALG-../ILV... | Calculate ord by alg in specified interleave |
| OAI/ID/ORDC../INTERLV...  OAI/ID/ORDC../ILV... | Set up spec. ORD in specified interleave |
| OAI/ID/CREATE/ALG-.... | Create empty pool of tpfdfindex-detailed file and build index according to string in alg |
| OAI/ID/RELEASE/ALG-... | Release pool of lowest level in tpfdf-index file structure and delete item/s in index |
| OAI/ID/PATH./ALG-... | Retrieve indexed file via specified path defined in dbtab |
| OAI/ID/CREATE/PATH./ALG- | Empty pool is created but  Only path is updated in index |

.

Examples

OAI/83 Displays Information about file with ID=83.

## OAI/DBTAB(DEF) – Display DBDEF table items

**FUNCTION:**

* Maintains and displays information from various TPFDF tables.
* The DBTAB parameter has three main purposes.
  + - * Rebuild the DBDEF tables
      * Display the contents of the DBDEF table
      * Maintain the ID restriction table.

**FORMAT:**

* Following entries can be entered as ZUDFM OAI/DBTAB/ or ZUDFM DEF.

|  |  |  |
| --- | --- | --- |
| **FOR DISPLAYING INFORMATION ABOUT ONE FILE** | | |
| DEF ID | * General information for given file | |
| DEF ID/COM | * Common subtable for given file | |
| DEF ID/DBG | * Tpfdf general subtable for given file | |
| DEF ID/DBK | * Display default keys for a specific file | |
| DEF ID/OPT | * Display SW02SR option bit meanings for specific file | |
| DEF ID/PGN | * Display program that contains DBDEF for the file | |
| DEF ID/IDX | * Displays TPFDF Index definition for the given file | |
| DEF ID/BRF | * Backreference route for given file | |
| **FOR DISPLAYING INFO ABOUT ALL FILES IN A GIVEN APPLICATION** | | |
| DEF APLq | * Displays general information * q is application character (Refer Section 2.6) | |
| DEF APLq/PGN | * Display Program Reference | |
| **DISPLAYS SHOWING INFO FOR ALL FILE** | | |
| DEF | Displays general information for all files | |
| DEF COM | Displays COMMON SUBTABLE FOR ALL FILES | |
| DEF DBG | General sub-table for all files | |
| DEF DBK | Key sub-table for all files | |
| DEF IDX | TPFDF index sub-table for all files | |
| DEF DBS | Database sub-table for all files | |
| DEF PGN | Program reference for all files | |
| DEF REC | Recoup info for all files | |
| **TPFDF SUPPORT ENTRIES** | | |
| DEF CNT | Subtable count display | |
| DEF INIT | Rebuild dbdef table | |
| DEF OPT/OPn | Display option bit settings for OPn | |
| DEF  OPT/OPn/MASK | Display files with matching OPn settings  where OPn is the option byte or cin  MASK is 8 Y/N/\* chars | |
| **MAINTAIN ID RESTRICTION TABLE|** | | |
| DEF WID/APLG | | Displays file ID for application type ‘G’ |
| DEF WID/ADD/GXXXXXX-XXXX | | Define range of IDs that are to be applied to application G |
| DEF WID/DIS | | displays ID restriction table |
| DEF WID/NEW/qyn | | Reserves next available ‘n’ IDs for Application ‘Q’ and Type ‘y’ |
| DEF WID/DID | | Displays IDs reserved by NEW |
|  | |  |

## OAINIT(INIT)– INITIALIZE FILES

**FUNCTION**:

* + - * Initializes the entire file or a part of a file.

**FORMAT:**

|  |  |
| --- | --- |
| * INIT ID/fileid | * Initialize specified ID * (fileid = 2-character or 4-hexadecimal file ID to be initialized) |
| * INIT ID/fileid FVN/ver | * Initialize specified ID and file version * (ver = 1-byte hexadecimal file version number) |
| * INIT ID/fileid |BOR - EOR * INIT ID/fileid/FVN-vn| BOR – EOR | * Initialize specified ID * From given range of Ordinal number from BOR to EOR |
| * INIT ID/fileid|BOR - LAST | * Initialize specified ID and file version * From given Beginning Ordinal (BOR) to the last ordinal |
| * INIT ID/fileid HOLD | * Initialize while holding records |
| * INIT ID/fileid NOHOLD | * Initialize without holding records |
| * INIT ID/fileid RELEASE | * Initialize and release forward chains |

**REQUIREMENTS & RESTRICTIONS:**

* The INIT Command needs to be entered TWICE since file is changed permanently when initialized. First time it results in warning and when entry is given second time, process of initialization is completed.
* The use of the ZUDFM INIT command should be restricted on the live system with the ZUDFM OAE command.

## OA(FAD) – DISPLAY THE CONTENTS OF A SUB-FILE

**FUNCTION**

Links to a sub-file and displays the contents.

**FORMAT:**

|  |
| --- |
| ZUDFM OA/ fileaddress OR  ZUDFM FAD fileaddress |

**DESCRIPTION:**

fileaddress Specifies the 4 byte file address of a DF file.

## OA\*(DISPLAY) – DISPLAY LRECs IN A SUB-FILE

**FUNCTION:**

* Displays the contents of LRECs in various formats using different types of search criteria.
* When issued without parameters, all the LRECs in the sub-file are displayed.

**FORMAT:**

There are 3 formats of this entry

* + - 1. ZUDFM OA \* [format parameters] [LREC Parameters]
      2. ZUDFM OA \*[format parameters] [Key parameters]
      3. ZUDFM OA \* [Block header parameters]

**FORMAT PARAMETERS:**

|  |  |
| --- | --- |
| **D** | Displays the file address and displacement of each LREC and the contents of each LREC in character format. |
| **H** | Displays the file address and displacement of each LREC and the contents of each LREC in hexadecimal format. |
| **L** | Displays the contents of each field in each LREC, with their DSECT labels.  This macro needs a macro label set which can be built using the MLS support |
| **T** | Display TLRECs (Technical LRECs) |

**LREC RANGE PARAMETERS**

|  |  |  |
| --- | --- | --- |
| lrecnum | sequence number of LREC number to be displayed | OA\*D3  Displays 3rd LREC |
| lrecnum-lrecnum | Specifies range of LREC number to be displayed | OA\*D3-5  Displays 3rd,4th and 5th LREC |
| lrecnum-LAST | Specifies the range from the LREC number to the last LREC | OA\*D4-LAST  Displays all LREC from 4th onwards |
| Combination of this parameters can be used  OA\*D3-5/8/12-LAST  Will Display 3rd, 4th, 5th (Described by 3-5)  8th (Described by 8)  all LRECs from 12th onward (Described by 12-LAST) | | |

**KEY PARAMETERS**

|  |  |  |
| --- | --- | --- |
| R Parameter   * + - D/NN Or     - L/CC | Parameter from the DSECT   * + - Displacement within LREC (Can’t be 0)     - Dsect label of displayed LREC | Ex   * + - D/12 (For referring 12th byte in LREC)     - L/CC (For referring CC field |
| **EITHER OF FOLLOWING 2 PARAMETER IS USED TO SPECIFY DATA TO BE COMPARED WITH** | | |
| S Parameter   * + - .S- | * + - Search value to be compared with the content of field specified by R parameter     - Max .length allowed is 49 characters | Ex.  .S-80|TEST  .S-|TEST |
| M Parameter   * .M- | * + - 1-byte Input Mask | Eg.  .M-80 (To check Bit 0 of the field specified in R parameter |
| **ORGANIZATION OF THE FILE** | | |
| * + - .O- | * + - To Specify the Organization of sub-file |  |
| **COMPARISON OPERATOR BETWEEN PARAMETERS SPECIFED IN R AND .S- or .m-** | | |
| * + - .C- | For .S-, one of the following is used   * + - GT (Greater Than)     - LT (Less than)     - NE (Not Equal)     - EQ (Equal –Default)     - NH (Not High)     - NL (Not Low)     - GE (Greater than or equal)     - LE (LE) | * + - Field specified by R is considered as 1st Operand and     - that by S is considered as 2nd operand   R -D/03.S-/|A.C-GT means that looking for LREC, in which 3rd byte of LREC is Greater than ‘A’ |
| * + - .C- | For M-, one of the following is used   * + - Z (All Zeros)     - O (All Ones)     - M (Mix of Ones and Zeros)     - NZ (Non-zeros     - NO (Not Ones)     - NM (Not Mixed) |  |

**BLOCK PARAMETERS**

|  |  |
| --- | --- |
| **D/HDR** | Displays the sub-file block header in character format |
| **H/HDR** | Displays the sub-file block header in hexadecimal format. |

**REQUIREMENTS & RESTRICTIONS:**

* + - * Before using this command the file must be linked by using the ZUDFM OA command.

## OAA(ADD) – ADD LRECS TO A SUB-FILE

**FUNCTION**

* Adds LRECs to the currently linked sub-file.
* The LRECs can be added to any position in the sub-file.
* Uses the designated keys to identify the location where an LREC can be added.
* The size of the LREC is dynamically calculated and inserted in the 2-byte field.

**FORMAT**

|  |  |
| --- | --- |
| * **PRIMARY KEYWORD** | |
| ZUDFM | |
| * **SECONDARY KEYWORD** | |
| OAA OR ADD | |
| * **PRIMARY KEYWORD** | |
| lrecnum | Sequence number that will be used for the added LREC |
| /lrecid | Primary Key |
| * Data content can be EBCDIC data or hexadecimal data * Use | to switch between these types of data * It is assumed that variable starts with hexadecimal | |

**Requirements & Restrictions:**

* Before using this command, the file must be linked using the ZUDFM OA command.
* Cannot be used with the sequence number when the file has default keys defined.
* TLRECs cannot be added using this command.
* ZUDFM uses a sequence counter to ensure the file integrity to avoid simultaneous updates.

**Example**

|  |  |
| --- | --- |
| * OAA/80|FWB|0000 | * Adds LREC with Key X’80’ * File has default key defined and LREC number is calculated automatically so that file organization is maintained |
| * OAA/001680|FWB|0000 | * Adds LREC with Key 80 and LengthX’16’ remaining fields after data provided will be padded by X’00’ |

## OAX(DELETE) – DELETE LRECS FROM A SUB-FILE

**FUNCTION**:

* Deletes the LRECs from a sub-file.

**FORMAT:**

|  |  |
| --- | --- |
| * **PRIMARY KEYWORD** | |
| ZUDFM | |
| * **SECONDARY KEYWORD** | |
| OAX or DELETE | |
| * **PRIMARY PARAMETER** | |
| lrecnum Or | Lrec Sequence number to be deleted |
| ALL | All LRECs in the sub-file are to be deleted |
| * **SECONDARY PARAMETER for lrecnum** | |
| **-LRecNum** | Ending LREC Number up to which LRECs need to be deleted |
| **-LAST** | If all the remaining LRECs needs to be deleted |
| **/lrecnum** | Used if LREC numbers to be deleted are not in sequence |

**REQUIREMENTS & RESTRICTIONS:**

* Before using this command, the file must be linked using the ZUDFM OA command.
* Sequence number of the LRECs can be found using displaying the sub-file. This sequence number has to be used while deleting the LREC.
* LREC sequence numbers starts from 1.
* The LREC ranges have to be in the ascending order. If one is found out of order, then all the subsequent LREC ranges are ignored.
* ZUDFM uses a sequence counter to ensure the file integrity to avoid simultaneous updates.
* Heavily modified databases may become difficult to maintain if the sequence counter is changed constantly.
* Utilities to locate hold and write a file can be developed by users to modify the file.

**EXAMPLE**

|  |  |
| --- | --- |
| * ZUDFM DELETE 1 | DELETE 1st LREC |
| * ZUDFM DELETE 1/4/7 | Delete 1st, 4th and 7th LREC |
| * ZUDFM OAX1-3/5 | Delete 1st to 3rd LREC and 5th LREC |
| * ZUDFM OAX 19-LAST | Delete all LREC starting from 19th LREC |
| * ZUDFM OAX ALL | Delete all the LRECs of the sub-file |

## OAR(REPLACE) – REPLACE USER DATA IN AN LREC

**FUNCTION:**

It replaces the user data portion of a LREC.

**FORMAT:**

|  |  |
| --- | --- |
| * **PRIMARY KEYWORD** | |
| ZUDFM | |
| * **SECONDARY KEYWORD** | |
| REPLACE or OAR | |
| * **PRIMARY PARAMETER** | |
| Lrecnum/ OR | Sequence number of the LREC |
| LBL- | * DSECT field to be replaced. (Do not include the first 4 character of the DSECT name). * When using this parameter with fixed length fields, the length of the fields must match the length of the field * For this, macro label set be build using Macro Label Set (MLS) support |
| * **SECONDARY PARAMETER FOR LBL-** | |
| ORG- | * Specifies the LREC ID. * Use this parameter when the label specified by the LBL parameter is not defined for the LREC ID of the record being replaced. * This parameter indicates the LREC ID for which the label is defined. |
| * With all these data content is entered. * Data content can be either character or hexadecimal * While transferring from hexadecimal to character and vice-versa, use | as delimiter to switch between them | |

**REQUIREMENTS & RESTRICTIONS:**

* The file should be linked and displayed before using this command.
* If the LREC is not replaced successfully, it has to be displayed again before using this command.
* TLRECs cannot be replaced by this command.
* LREC length field should not be entered.
* SIZE field in the LREC should not be changed directly. TPFDF adjusts the size by itself.
* ZUDFM uses a sequence counter to ensure the file integrity to avoid simultaneous updates.

**EXAMPLE**

* **OAR2/80|FWB|0000|FX216**

Replaces 2nd LREC content with provided data

* **OAR2/LBL-RST/20|ORG-20**

## OAF(FCH) – DISPLAY FORWARD & BACKWARD CHAINING

**FUNCTION:**

* Displays the forward and backward chaining of a sub-file or a file address.
* If issued without parameters, chaining is displayed beginning with the file address that is currently linked.
* When issued with a specific file address, the file address becomes the currently linked file.

**FORMAT:**

|  |  |
| --- | --- |
| ZUDFM | PRIMARY KEYWORD |
| OAF or  FCH | SECONDARY KEYWORD |
| fileaddr | PRIMARY PARAMETER  specifies 8 or 16- character hexadecimal file address |

**INFO**

* Outputs from the ZUDFM OAF command stops after 500 chains have been displayed.
* Block Trailer and Command information is not displayed under following condition
  + - &ACPDBFS variable is set to 0 in the DBLCL macro.
    - The block trailer size of the file is less than 36 bytes.
    - Nodes are being displayed in a B+Tree index file.

## OAP(PACK) – PACK A SUB-FILE

**FUNCTION:**

* Removes unused space in a sub-file.
* Builds or rebuilds a B+ Tree index.
* Can be used to pack the LRECs into a minimum number of blocks regardless of the current packing density in the sub-file and whether the LRECs have been deleted or not.
* Blocks are packed to the Packing Limit Percentage specified in DBDEF Macro (PLI)

**FORMAT:**

|  |  |
| --- | --- |
| ZUDFM | PRIMARY KEYWORD |
| OAP Or  PACK | SECONDARY KEYWORD |

**REQUIREMENTS & RESTRICTIONS:**

* The file must be linked before using this command.
* The file is held during packing the file.

## OAL(LOG) – DISPLAY RECENT OA COMMANDS

**FUNCTION:**

* Displays the recently used ZUDFM OA commands, which perform modification.
* The display is built from the modification log that TPFDF maintains.

**FORMAT:**

|  |  |
| --- | --- |
| ZUDFM | PRIMARY KEYWORD |
| OAI OR LOG | SECONDARY KEYWORD |

**RESPONSE**

UDFM0421I OA MODIFICATION LOG DISPLAY

DATE TIME CRT SI/DU FILEADDR ID ENTRY

15JAN 07.58 760310 JY/SL 84F4B63B 7 D OAX62

15JAN 07.51 760310 JY/TS 0FE00063 9 T OAX433

## OAE(RESTRICT) – RESTRICT TABLE HANDLING

**FUNCTION**:

* + - * Used to inspect and edit the table of restricted commands.
      * Used to specify the terminals from which certain ZUDFM commands can be entered.

**FORMAT:**

|  |  |
| --- | --- |
| **PRIMARY KEYWORD** | |
| ZUDFM | |
| **SECONDARY KEWORD** | |
| OAE Or  RESTRICT | |
| **PRIMARY PARAMETER** | |
| * + - A     - X     - DISPLAY | * To Add entry to table * To Delete entry from table * To Display restricted table |
| **SECONDARY PARAMETER TO A or X** | |
| * + - TERMINAL | * + - LNIATA to be added to/deleted from table |
| * + - COMMAND | * + - Command to restricted or from which restriction needs to removed     - Don’t enter ZUDFM portion of command |
| **SECONDARY PARAMETER TO DISPLAY** | |
| * + - Command | display |
| * + - \* | DISPLAY FULL RESTRICTION TABLE |

**NOTES**.

* + - * INIT parameter of ZUDFM DEF command can’t be restricted by this command. This is done to make sure that Entry restriction table can be re-initialized if there is any problem.
      * You must be in CRAS state before using this command
      * A ZUDFM command that is listed in the entry restriction table can be entered only from the terminals specified.

**EXAMPLE**

* **ZUDFM RESTRICT A/710610|DELETE**

It restricts delete entry to 710610. Means that only 710610 can enter this entry

* **ZUDFM RESTRICT DISPLAY DELETE**

It display the list of terminal on which DELETE can be entered

* **ZUDFM RESTRICT DISPLAY \***

Displays the entry restriction table

## OAZ(TOD) – CONVERT TIME

**FUNCTION:**

It displays time and date information based on a 4- to 16-character hexadecimal time stamp.

**FORMAT:**

|  |  |
| --- | --- |
| * + - **PRIMARY KEYWORD** | |
| ZUDFM | UTILITY NAME |
| * + - **SECONDARY KEYWORD** | |
| OAZ or TOD | Function Name |
| * + - PARAMETER | |
| timevalue | 4-16 hexadecimal character time stamp |

# TPFDF QUICK REFERENCE TABLES



This section will discuss about the TPFDF Quick References.

Following terms will be discussed in this section

* TPFDF Basic Dumps
* Table of TPFDF Algorithm
* SW00RTN conditions
* TPFDF Option Byte Settings
* Meaning of the Global indicators
* ZUDFM Entries for Online maintenance of the file

## TPFDF Basic Dumps

| **Dump Number** | **DESCRIPTION** |
| --- | --- |
| DB0100 | FINWC/FIWHC Error on data level |
| DB0101 | Illegal item size found in item |
| DB0102 | Face error return |
| DB0104 | Illegal NAB found in block |
| DB0105 | Invalid TPFDB request (sequence) |
| DB0106 | R3 not loaded on SW00SR |
| DB0107 | FINWC error on level D |
| DB0108 | FILNC I/O error on data level |
| DB0109 | Too many files opened concurrently |
| DB010A | File ID not found in DBTAB |
| DB010B | FMSG error |
| DB010C | File update without hold option in open |
| DB010D | FILNC on SW00LVL with bad nab |
| DB010E | No block found after attac |
| DB010F | BAD block – NAB disagrees |
| DB0110 | Create not first command |
| DB0111 | TAPE = Parameter missing in Tapelog |
| DB0112 | File open at exit time |
| DB0115 | REF = was not found serrc exit |
| DB0116 | Invalid algorithm value defined |
| DB0117 | Add with default keys can not be resolved |
| DB0118 | File ID not found in DBTAB (taperead) |
| DB0119 | Bad block size found on tape (taperead) |
| DB011A | Level D not allowed |
| DB011B | Find error level D for rel.file(RELFC) |
| DB011C | Invalid addr in FADDR= |
| DB011E | Space=/spaceb= exceeds one phys.blk |
| DB011F | Measurement not active |
| DB0120 | Illegal chain loop |
| DB0121 | Key specification incorrect |
| DB0123 | Index/deindex on read-only path |
| DB0124 | Cmd/param. Not allowed for p-type file |
| DB0125 | P-type add block but no fch defined |
| DB0126 | TPFDB UKEY unique key value exceeded |
| DB0127 | Index LREC size too big, exceeds SPACE= |
| DB0128 | Replace command with TAPE= is invalid |
| DB012D | New style retain – GW06SR in error |
| DB0130 | No space in CTIN for DBTAB table |
| DB0131 | Duplicate ID in DBDEF table |
| DB0132 | Tape write SYNCC error |
| DB0133 | Error in size for DBCPY P-Type file |
| DB0134 | MDBF information mismatch |
| DB0135 | Invalid SSU count |
| DB0136 | Invalid exclude item count |
| DB0137 | Excluded SSU attempting file access |
| DB0138 | Accessing already released pool address |

## TABLE OF TPFDF ALGORITHMS

| **ALGORITHM NAME** | **ALGORITHM LENGTH & TYPE** | **ADDRESSABLE ORDINALS** |
| --- | --- | --- |
| #TPFDB01 | 1 Character alphabet (A-Z) | 26 Ordinals (0-25) |
| #TPFDB02 | 2 Characters alphabet (AA-ZZ) | 676 Ordinals (0-675) |
| #TPFDB03 | 3 Characters alphabets (AAA-ZZZ) | 17576 Ordinals (0-17575) |
| #TPFDB04 | No Algorithm to be Used | Only one ordinal number used |
| #TPFDB05 | User supplies ordinal number in 4 byte field | As defined in ‘FACE’ for this record type |
| #TPFDB06 | 1 Byte alpha-numeric value (0-9,A-Z) | 36 ORDINALS (0-35) |
| #TPFDB07 | 2 Byte alpha-numeric value  (AA-AZ…A0..A9..ZA..ZZ.. Z0..Z9..00.09..0A..0Z..9A.9Z) | 1296 ORDINALS (0-1295) |
| #TPFDB08 | 3 Byte alpha-numeric value  (AAA..AAZ..AA0..AA9..A00..A09..Z90…999..ZZZ) | 46656 ORDINALS (0-46655) |
| #TPFDB09 | 8 BYTE VALUE (HASHED) | As defined in ‘FACE’ for this record type |
| #TPFDB0A | 1 Byte alpha-numeric or special character ( . , $ ,\* , - , / , # , @ ) | 43 Ordinals (0-42) |
| #TPFDB0B | 2 Byte alpha-numeric or special characters | 1849 Ordinals (0-1848) |
| #TPFDB0C | User supplied ordinal number in 2 byte field | As defined in ‘FACE’ for this record type |
| #TPFDB0D  (For use with fixed length LRECS) | Depend upon the LREC number in the file | As defined in ‘FACE’ for this record type |
| #TPFDB0F | For PARTITIONED files  10 byte string | As defined in ‘FACE’ for this record type |
| #TPFDBFF | FOR INDEXED Detail files | This algorithm is only for pool files and hence no ordinals involved |

## SW00RTN Conditions returned from TPFDF Commands

|  |  |
| --- | --- |
| **SW00RTN Return code** | **Condition encountered** |
| #BIT0 ON | I/O Error |
| #BIT1 ON | LREC Not Found |
| #BIT2 ON | FACE return error |
| #BIT3 ON | ALGORITHM error |
| #BIT4 ON | Data in Block is Corrupted |
| #BIT5 ON | EOF During FULLFILE Processing |
| #BIT6 ON | Sequence error using DBRST macro |
| #BIT7 ON | Sort/Merge error |

TPFDF has defined equates to test SW00RTN Values

|  |  |
| --- | --- |
| **Return value SW00RTN** | **Condition** |
| #TPFDBEX (X'AB') | Any Serious error except indexing errors |
| #TPFDBER (X'BB') | Any Serious error |
| #TPFDBNR (X'44') | LREC not found |
| #TPFDBOK (X'00') | No error , Record found |

TPFDF Provides short form of testing SW00RTN using SPM's as follows

|  |  |
| --- | --- |
| **SPM EQUATE** | **PURPOSE** |
| DBFOUND(YES|NO) | Checks if LREC is found |
| DBERROR (YES|NO) | Check if Error occurred |
| DBEOF (YES|NO) | Check End of File Conditiion |
| DBIDX (YES|NO | Check for INDEX error |

## TPFDF OPTION BYTE SETTINGS

*SW00OP1 Settings :*

|  |  |
| --- | --- |
| **Bit Number** | **Implication** |
| BIT0 | Implement Full backward chaining ,  Needs to be set up for DBRED BACKWARD |
| BIT1 | Implement Automatic chain correction ,  Truncates chain if TPFDF finds broken chain ,  Data may be lost if set |
| BIT2 | Implement Current files ,  Set SW00NOC ,  wraps around LRECs and keeps only the latest data , can be set for log files |
| BIT3 | Implement Push down chaining , always adds new LRECs to the prime block |
| BIT4 | Issue checkpoint for each DBUKY macro ,  A mechanism to ensure serialization of adding related LRECs |
| BIT5 | If set SW00WRS is the size of prime block ,  SW00WRS size is first taken for overflow block and should it exceed the size converts it into a SW00ARS sized block |
| BIT6 | Checks for packing value always regardless of a DBDEL macro |
| BIT7 | Maintains file sequence update counter |

*SW00OP2 Settings :*

|  |  |
| --- | --- |
| **Bit Number** | **Implication** |
| BIT0 | Validates NAB every time a block is filed , Processing overhead involved |
| BIT1 | New pool blocks used for files after packing |
| BIT2 | New pool blocks are used for restore and old blocks are released |
| BIT3 | New pool blocks are used for tape load and old blocks are released |
| BIT4 | To be able to update a file using a FINWC and without a hold , No DB010C dump issued |
| BIT5 | Same as above but for overflow blocks |
| BIT6 | Issue a DB010C dump when file is modified without issuing a HOLD |
| BIT7 | Reserved for future use |

SW00OP3 Settings :

|  |  |
| --- | --- |
| **Bit Number** | **Implication** |
| BIT0 | Set up for Extended LRECs |
| BIT1 | Reserved for Future use |
| BIT2 | Reserved for Future use |
| BIT3 | Fixed file with index files referring to it |
| BIT4 | Checkpoint each time an Index LREC is added or deleted |
| BIT5 | Set this file for B+tree indexing |
| BIT6 | Default open with DETAC mode on the file (Expensive operation) |
| BIT7 | Set this up if the Unique Key feature is to be used |

## MEANING OF THE GLOBAL INDICATORS

|  |  |  |
| --- | --- | --- |
| **FIELD** | **NEEDED** | **FUNCTION** |
| SW00WID | Yes | * Specifies the file ID, which is a 2-byte value in the standard header of every block. * Each DSECT should have a unique ID. |
| SW00WRS | Yes | * Specifies the size of the prime block used in the file. |
| SW00ARS | No | * Specifies the size of the overflow blocks to be used in the file. * If not assigned the size is taken to be the same as WRS. |
| SW00BOR | Yes(F) | * Needed only for fixed and miscellaneous files. * It is used to specify the beginning ordinal for the file. * For a fixed file, always set the base ordinal number to zero. If this is a miscellaneous file, specify the symbol (usually defined in SYSEQ) that defines where the miscellaneous file starts in the miscellaneous fixed file type. |
| SW00EOR | Yes(F) | * Needed only for fixed and miscellaneous files. * It is used to specify the ending ordinal for the file. * For a fixed file, non-partitioned, always set the base ordinal number to -1. If this is a miscellaneous file, specify the symbol (usually defined in SYSEQ) that defines where the miscellaneous file starts in the miscellaneous fixed file type. * For a partitioned file, specify the number of prime blocks in each partition, not the total number of prime blocks in all partitions. |
| SW00ILV | No | * The number of interleaves. * Defaults to zero, meaning no interleaves. |
| SW00PTN | No | * The number of partitions that are there. * Defaults to zero, meaning no partitions. |
| SW01E0# | Yes(F) | * &SW01EO# specifies the FACE-type end ordinal number. * For non-partitioned & non-interleaved, set value to –1. * For partitioned or interleaved, set it to the total number of prime blocks in the file. |
| SW02FIL | Yes | * Set the DSECT macro name. |
| SW00NLR | No | * Set only when file used algorithm #TPFDB0D. * Specifies the number of fixed-length LRECS in the record. |
| SW00NOC | No | * Specifies the number of blocks to use in implanting the current file. Assign only when OP1 bit 2 is set; this limits the number of overflow blocks added as overflows. |
| SW00RBV | Yes | * Defines the algorithm used in this file. * For detail pool file, set it to #TPFDBFF. |
| SW00RCT | Yes(F) | * Specifies the file’s record type as defined in the Face Table. |
| SW00PIN | No | * Specifies a 0-100 value which will be used as the packing threshold. |
| SW00REF | Yes(T) | * This is valid only for T-type files. * Specifies the reference name of the W-type file that contains the T-type File. Defaults to GW01SR. |
| SW00SKE | No | * Specifies that Block Indexing Support is used. * Set this equal to the length in bytes, of the key fields to be removed from the first LREC of the overflow blocks for TLRECS. |
| SW00TQK | No | * Specifies the highest technical LREC. * For block indexing should be more than 2. * For B+ support should be more than 4. |
| SW00TYP | No | * Determines the type of the file. * If omitted taken from the 2-char of the file name. |

## ZUDFM ENTRIES FOR ONLINE MAINTENANCE OF TPFDF FILES

|  |  |  |
| --- | --- | --- |
| **OLD FORMAT** | **NEW FORMAT** | **DESCRIPTION** |
| OAI/DBTAB | ZUDFM DEF | Display Central DBDEF table |
| OAI/ | ZUDFM DEF | Initialise OAI Work File |
| OA/FADDR | ZUDFM FAD | Initialise specifying file address |
| OA\* | ZUDFM DISPLAY | TPFDF File display |
| OAI/DBTAB/DSECT/REC  OAI/DBTAB/DSECT/DBG OAI/DBTAB/DSECT/REC  OAI/DBTAB/DSECT/IDX  OAI/DBTAB/DSECT/PGN  OAI/DBTAB/DSECT/COM | ZUDFM DEF | Display DBDEF for DSECT |
| OAA | ZUDFM ADD | Add an LREC |
| OAF | ZUDFM FCH | Display chains |
| OAINIT | ZUDFM INIT | Initialise TPFDF File (Dangerous entry) |
| OAR | ZUDFM REPLACE | Replace an LREC |
| OAX | ZUDFM DELETE | Delete an LREC |
| OAS | ZUDFM TRACE | Use with SST for display of SW00SR and Blocks |
| OAZ | ZUDFM TOD | TOD-Clock conversion |
| OAH | ZUDFM HELP | TPFDF Online utility help display |