# VMS File Definition Language Facility Manual

Order Number: AA-LA81A-TE

**April 1988**

This document describes the VMS File Definition Language (FDL) Facility.

| | |
|---|---|
| **Revision/Update Information:** | This manual supersedes the *VAX/VMS File Definition Language Facility Reference Manual,* Version 4.4. |
| **Software Version:** | VMS Version 5.0 |

**digital equipment corporation**
**maynard, massachusetts**

**April 1988**



The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

| | | |
|---|---|---|
| DEC | DIBOL | UNIBUS |
| DEC/CMS | EduSystem | VAX |
| DEC/MMS | IAS | VAXcluster |
| DECnet | MASSBUS | VMS |
| DECsystem–10 | PDP | VT |
| DECSYSTEM–20 | PDT | |
| DECUS | RSTS | |
| DECwriter | RSX | digital™ |

ZK4544

**HOW TO ORDER ADDITIONAL DOCUMENTATION**
**DIRECT MAIL ORDERS**

| USA & PUERTO RICO* | CANADA | INTERNATIONAL |
|---|---|---|
| Digital Equipment Corporation<br>P.O. Box CS2008<br>Nashua, New Hampshire<br>03061 | Digital Equipment<br>of Canada Ltd.<br>100 Herzberg Road<br>Kanata, Ontario K2K 2A6<br>Attn: Direct Order Desk | Digital Equipment Corporation<br>PSG Business Manager<br>c/o Digital's local subsidiary<br>or approved distributor |

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

## Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

---

™ PostScript is a trademark of Adobe Systems, Inc.

# Contents

## Contents

## TABLES

# Preface

## Intended Audience

This manual is intended for all programmers using VMS RMS data files. This audience includes high-level language programmers who use only their language's input/output statements.

## Document Structure

This document consists of the following sections:

- Description—Provides an overview and detailed usage information about the File Definition Language, Edit/FDL Utility, and Create/FDL Utility.
- Usage Summary—Describes how to invoke, exit, and direct information from the Edit/FDL and Create/FDL Utilities.
- CREATE/FDL Qualifiers—Describes the /LOG qualifier available when you invoke the Create/FDL Utility.
- EDIT/FDL Qualifiers—Describes the qualifiers available when you invoke the Create/FDL Utility.
- EDIT/FDL Commands—Describes the Edit/FDL Utility commands.
- Examples—Provides additional examples of common operations that you perform with the Edit/FDL Utility.

## Associated Documents

To use the File Definition Language Facility, you should be familiar with the following manuals:

- *Guide to VMS File Applications*
- *VMS Analyze/RMS_File Utility Manual*
- *VMS Convert and Convert/Reclaim Utility Manual*
- *VMS Record Management Services Manual*

## Conventions

| Convention | Meaning |
|---|---|
| RET | In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.) |
| CTRL/C | A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box. |
| $ SHOW TIME<br>05-JUN-1988 11:55:22 | In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red. |
| $ TYPE MYFILE.DAT<br>.<br>.<br>. | In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown. |
| input-file, . . . | In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted. |
| [logical-name] | Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.) |
| quotation marks<br>apostrophes | The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark. |

# New and Changed Features

Version 5.0 of the File Definition Language (FDL) Facility includes the following new features:

- A new primary attribute, NETWORK, lets you set run-time network characteristics.
- A new secondary KEY attribute, COLLATING_SEQUENCE, lets you specify a collating sequence in a National Character Set (NCS) library.
- Two new KEY TYPES support the use of NCS collating sequences:
  - COLLATED
  - DCOLLATED
- A new secondary FILE attribute, FILE_MONITORING, has been added.

# FDL Description

The File Definition Language (FDL) Facility is comprised of the File Definition Language, the Create/FDL Utility, and the Edit/FDL Utility.

The description of the facility is divided into three parts.

The first part describes the FDL primary and secondary attributes. The second part explains how to create FDL files, primarily by using EDIT/FDL. The third part describes how FDL files are used by the VMS RMS utilities and callable routines.

## 1 File Definition Language

File design is one of the most important parts of efficient data processing, and the File Definition Language (FDL) helps you define specifications for data files. Section 1.1 gives a brief overview of the primary and secondary attributes. Then each primary attribute is listed in alphabetical order with detailed explanations of their secondary attributes.

### 1.1 FDL Primary and Secondary Attributes

An FDL file consists of a collection of file *attributes* grouped into related sections. The 14 section headings are called *primary attributes* and must be specified in the following order:

- TITLE
- IDENT
- SYSTEM
- FILE
- DATE
- RECORD
- ACCESS
- NETWORK
- SHARING
- CONNECT
- AREA
- KEY
- ANALYSIS_OF_AREA
- ANALYSIS_OF_KEY

The TITLE, IDENT, AREA, KEY, ANALYSIS_OF_AREA, and ANALYSIS_OF_KEY sections take values.

The SYSTEM, FILE, DATE, RECORD, ACCESS, SHARING, and CONNECT sections do not take values. They do, however, serve as labels for the sections.

The ANALYSIS_OF_AREA and ANALYSIS_OF_KEY sections appear only in FDL files created with the Analyze/RMS_File Utility.

Attributes within a section are called *secondary attributes*. Certain secondaries can have a third level of attributes called *qualifiers*. A completed FDL file consists of attribute keywords followed by their assigned values. Lowercase letters are legal anywhere; they are equivalent to uppercase letters.

The description of these attributes and the secondary attributes contains cross-references to the fields (parameters) of the VMS RMS control blocks.

The value assigned to an attribute must be one of the following types:

| | |
|---|---|
| Switch | Is a logical value, set to TRUE (YES) or FALSE (NO). TRUE or YES sets the attribute; FALSE or NO clears it. You can specify TRUE, YES, FALSE, and NO as T, Y, F, and N. |
| Keyword | Is an actual word that you must type after the attribute name. You can truncate a keyword to its unique characters. |
| String value | Is a character string that you must type after the attribute name. The null string is a valid string value. You should enclose a string value in a pair of single or double quotation marks. |
| Number | Is a decimal number. |

Throughout this description, the term "DECnet operations" refers to remote file access between two VMS operating systems. Unless stated otherwise, attributes are supported for DECnet operations.

### 1.1.1 ACCESS Section

The ACCESS section allows you to specify the file processing operations you want performed on your file. The ACCESS keyword itself takes no values. Table FDL–1 lists the ACCESS secondary attributes and their default values.

**Table FDL–1 Default Values for ACCESS Secondaries**

| Secondary | Default Value |
|---|---|
| BLOCK_IO | FALSE |
| DELETE | FALSE |
| GET | GET when performing an Open service |
| PUT | PUT when performing a Create service |
| RECORD_IO | FALSE |
| TRUNCATE | FALSE |
| UPDATE | FALSE |

#### *BLOCK_IO*

Is a switch specifying that block I/O operations involving Read or Write RMS services are to be performed, depending on whether you have specified the GET (Read service) or the PUT (Write service) ACCESS secondary attributes. If you specify BLOCK_IO, no record I/O operations (such as DELETE, GET, PUT, TRUNCATE, or UPDATE) can be performed. This secondary also allows you to use the Space service.

This attribute corresponds to the FAB$B_FAC field, the BIO option.

### DELETE
Is a switch allowing Delete RMS operations to be performed.

This attribute corresponds to the FAB$B_FAC field, the DEL option.

### GET
Is a switch allowing Get or Find RMS services. GET is the default when you are opening the file and when one of the following conditions exists:

- No other ACCESS section secondary attribute is defined.
- The DELETE or UPDATE secondary attributes in the SHARING section have been specified.

If you also specify the BLOCK_IO attribute, you may perform Read services.

This attribute corresponds to the FAB$B_FAC field, the GET option.

### PUT
Is a switch allowing Put or Extend RMS services. PUT is the default when you are creating a file. If you also specify the BLOCK_IO attribute, you can perform Write services.

This attribute corresponds to the FAB$B_FAC field, the PUT option.

### RECORD_IO
Is a switch allowing mixed record I/O and block I/O operations under certain circumstances (see the *VMS Record Management Services Manual* for more information).

This attribute corresponds to the FAB$B_FAC field, the BRO option.

### TRUNCATE
Is a switch allowing Truncate RMS operations.

This attribute corresponds to the FAB$B_FAC field, the TRN option.

### UPDATE
Is a switch permitting Update or Extend RMS services.

This attribute corresponds to the FAB$B_FAC field, the UPD option.

## 1.1.2 ANALYSIS_OF_AREA Section
The ANALYSIS_OF_AREA section is created and supplied with values by the Analyze/RMS_File Utility. This section appears only in FDL files that describe indexed files.

This primary section has only one secondary—RECLAIMED_SPACE.

### RECLAIMED_SPACE
ANALYZE/RMS_FILE supplies a number value for this secondary. The value is the number of blocks in the area that were reclaimed with the Convert Utility (using the /RECLAIM qualifier). For more information about using CONVERT/RECLAIM, see the *VMS Convert and Convert/Reclaim Utility Manual.*

### 1.1.3 ANALYSIS_OF_KEY Section

The ANALYSIS_OF_KEY section is created and supplied with values by the Analyze/RMS_File Utility. It appears only in FDL files that define an indexed file.

The Edit/FDL Utility uses the ANALYSIS_OF_KEY section in its Optimize script.

The primary attribute ANALYSIS_OF_KEY has a value that is the number of the key being analyzed (0 is the primary key).

Table FDL–2 lists the ANALYSIS_OF_KEY secondary attributes. All values returned to the attributes are of the numerical type.

**Table FDL–2 Default Values for ANALYSIS_OF_KEY Secondaries**

| Secondary | Default Value |
|---|---|
| DATA_FILL | None |
| DATA_KEY_COMPRESSION | None |
| DATA_RECORD_COMPRESSION | None |
| DATA_RECORD_COUNT | None |
| DATA_SPACE_OCCUPIED | None |
| DEPTH | None |
| DUPLICATES_PER_SIDR | None |
| INDEX_COMPRESSION | None |
| INDEX_FILL | None |
| INDEX_SPACE_OCCUPIED | None |
| LEVEL1_RECORD_COUNT | None |
| MEAN_DATA_LENGTH | None |
| MEAN_INDEX_LENGTH | None |

#### *DATA_FILL*

Shows the percentage of bytes per bucket in the data level that has been filled.

#### *DATA_KEY_COMPRESSION*

Shows the percentage of compression that has occurred in the primary keys. If the keys added up to 1000 bytes and compression reduced that figure to 600 bytes, the value shown in the DATA_KEY_COMPRESSION attribute would be 40 (for 40%).

Negative compression may occur because of the overhead involved. If you see a negative value, you should disable that type of compression in the KEY section.

#### *DATA_RECORD_COMPRESSION*

Is the percentage of compression that has occurred in the level 0 data record. If the data records added up to 100,000 bytes and compression reduced that figure to 70,000 bytes, the value shown in the DATA_RECORD_COMPRESSION attribute would be 30 (for 30%).

Negative compression may occur because of the overhead involved. If you see a negative value, you should disable that type of compression in the KEY section.

This attribute applies only to the primary key.

### *DATA_RECORD_COUNT*
Shows the number of data records.

### *DATA_SPACE_OCCUPIED*
Shows the size in blocks of the level 0 of the index structure.

### *DEPTH*
Shows the number of index levels in the index structure. The value does not include the data level.

### *DUPLICATES_PER_SIDR*
Shows the average number of duplicate key values for the secondary index data records (SIDR); that is, the value is the total number of duplicates divided by the total number of SIDRs.

This attribute applies only to alternate keys.

### *INDEX_COMPRESSION*
Shows the percentage of compression that has occurred in the index records within the index levels. If the full indexes amounted to 10,000 bytes and compression reduced this value to 8000 bytes, the value shown in the INDEX_COMPRESSION attribute would be 20 (for 20%).

### *INDEX_FILL*
Shows the percentage of bytes per bucket that have been filled in the index levels.

### *INDEX_SPACE_OCCUPIED*
Shows the size in blocks of the index levels (level 1 and greater).

### *LEVEL1_RECORD_COUNT*
Indicates the number of records in the level 1 index, which is the index level immediately above the data. When duplicate key values (for SIDRs) have been specified, even when SIDR overflow buckets exist, the tuning algorithm of EDIT/FDL is made more accurate.

Generally, every bucket on level 0 of an alternate key has a pointer record from level 1 of that alternate key. However, there are no pointers from level 1 to any overflow buckets. LEVEL1_RECORD_COUNT keeps track of how many records are in level 1, particularly when duplicate key values force overflow buckets to be created.

### *MEAN_DATA_LENGTH*
Shows the average length in bytes of the data records. This does not take compression into account.

### *MEAN_INDEX_LENGTH*
Is the average length in bytes of the index records. This does not take compression into account.

### 1.1.4 AREA Section

The AREA section is an RMS-specific region of an indexed file. You cannot create or manipulate these areas from a high-level programming language. Instead, VMS RMS automatically creates various areas for you when you create an indexed file.

If you want to create or manipulate areas in an indexed file, you must include the AREA primary attibute in an FDL file. The AREA primary acts as a header for a section in the FDL file that describes areas. It takes a value that must be a number in the range 0 to 254. The number identifies the area. To define multiple areas for an indexed file, you must specify a separate AREA section for each area.

Most AREA secondaries (except EXACT_POSITIONING, POSITION, and VOLUME) have corresponding FILE secondaries. Any values you specify for these AREA secondaries override any you specify for the corresponding secondaries in the FILE section.

This attribute corresponds to the XAB$B_AID field.

Table FDL–3 lists the AREA secondary attributes and their default values.

**Table FDL–3 Default Values for AREA Secondaries**

| Secondary | Default Value |
|---|---|
| ALLOCATION | 0 |
| BEST_TRY_CONTIGUOUS | FALSE |
| BUCKET_SIZE | 0 |
| CONTIGUOUS | FALSE |
| EXACT_POSITIONING | FALSE |
| EXTENSION | 0 |
| POSITION | None |
| VOLUME | 0 |

#### *ALLOCATION*

Sets the number of blocks that you will initially allocate for this area. Its value must be an integer in the range 0 to 4,294,967,295. The default is 0, which means that the system will not allocate space for this area.

This attribute corresponds to the XAB$L_ALQ field.

#### *BEST_TRY_CONTIGUOUS*

Is a switch that controls whether the area is to be allocated contiguously if there is enough space for it. If you set the switch to YES and there is enough space for the area, the area is allocated contiguously. If you set the switch to YES and there is not enough space, the area is allocated noncontiguously.

If you set the switch to the default, NO, this attribute has no effect.

This attribute corresponds to the XAB$B_AOP field, the CBT option.

#### *BUCKET_SIZE*

Sets the number of blocks per bucket for this area. Its value must be an integer in the range 0 to 63. The default value is 0, which means that VMS RMS calculates the smallest bucket size capable of holding the largest record.

If the file is to be processed by RMS–11, the bucket size is limited to 32 blocks.

This attribute corresponds to the XAB$B_BKZ field.

## CONTIGUOUS

Is a switch that controls whether the file must be allocated contiguously.

When you set the switch to YES, this attribute means that the area must be allocated contiguously. If there is not enough contiguous space for the area, you receive an error when you try to create the data file.

When you set the switch to the default, NO, this attribute is ignored.

This attribute corresponds to the XAB$B_AOP field, the CTG option.

## EXACT_POSITIONING

Is a switch, set by default to NO. When you set this switch to YES, then the exact positioning of the area you specified with either the POSITION CYLINDER or the POSITION LOGICAL attribute must take place successfully, or else an error occurs.

When you set the switch to NO, the system positions the area as close as possible to the location requested.

This attribute corresponds to the XAB$B_AOP field, the HRD option.

## EXTENSION

Sets the number of blocks for the default extension quantity for the area. The extension is the amount of space that the system adds to the area when the area is filled up.

The value must be an integer in the range 0 to 65,535. The default is 0, which means that the extension size is determined by the system whenever the area requires extending.

This attribute corresponds to the XAB$W_DEQ field.

## POSITION

The POSITION attribute controls the positioning of the area. Its value must be one of the following keywords:

| | |
|---|---|
| ANY_CYLINDER | Begins the area on any cylinder boundary. |
| CYLINDER | Begins the area on the boundary of the cylinder specified by number. |
| FILE_ID | Places the area as close to the specified file as possible. The file must exist. The value you specify must be a valid file ID containing the file identification number, the file sequence number, and the relative volume number. It has the following form (parentheses included):<br><br>(FID-num,FID-seq,RVN)<br><br>This attribute is not supported for DECnet operations; NONE is used. |

| | |
|---|---|
| FILE_NAME | Places the area as close to the specified file as possible. The file must exist. The value you specify must be a valid file specification. This attribute is not supported for DECnet operations; NONE is used. |
| LOGICAL | Begins the area at a logical block, specified by number. |
| NONE | Means that you do not want to control the placement of the area. NONE is the default value. |
| VIRTUAL | Begins the area at a virtual block, specified by number. |

The POSITION attribute corresponds to the XAB$B_ALN, XAB$L_LOC, and XAB$W_RFI fields.

### *VOLUME*

Specifies the relative number of the volume in a Files–11 disk volume set on which the area is to reside.

This value must be an integer in the range 0 to 255.

The default is 0, which means that you do not want to control the volume placement of the area.

This attribute corresponds to the XAB$W_VOL field.

## 1.1.5 CONNECT Section

The CONNECT section specifies run-time attributes that are application-dependent and related to record access and performance. The CONNECT keyword itself takes no values. Table FDL–4 lists the CONNECT secondary attributes.

**Table FDL–4 Default Values for CONNECT Secondaries**

| Secondary | Default Value |
|---|---|
| ASYNCHRONOUS | None |
| BLOCK_IO | None |
| BUCKET_IO | None |
| CONTEXT | None |
| END_OF_FILE | None |
| FAST_DELETE | None |
| FILL_BUCKETS | None |
| KEY_GREATER_EQUAL | None |
| KEY_GREATER_THAN | None |
| KEY_LIMIT | None |
| KEY_OF_REFERENCE | None |
| LOCATE_MODE | None |
| LOCK_ON_READ | None |
| LOCK_ON_WRITE | None |
| MANUAL_UNLOCKING | None |
| MULTIBLOCK_COUNT | None |

**Table FDL–4 (Cont.) Default Values for CONNECT Secondaries**

| Secondary | Default Value |
|---|---|
| MULTIBUFFER_COUNT | None |
| NOLOCK | None |
| NONEXISTENT_RECORD | None |
| READ_AHEAD | None |
| READ_REGARDLESS | None |
| TIMEOUT_ENABLE | None |
| TIMEOUT_PERIOD | None |
| TRUNCATE_ON_PUT | None |
| TT_CANCEL_CONTROL_O | None |
| TT_PROMPT | None |
| TT_PURGE_TYPE_AHEAD | None |
| TT_READ_NOECHO | None |
| TT_READ_NOFILTER | None |
| TT_UPCASE_INPUT | None |
| UPDATE_IF | None |
| WAIT_FOR_RECORD | None |
| WRITE_BEHIND | None |

## *ASYNCHRONOUS*

Is a switch indicating that I/O operations are to be performed asynchronously. When you specify this attribute, VMS RMS returns control to your program as soon as an I/O operation is initiated, even though that operation may not yet be completed. The ASY is ignored for process permanent files.

This attribute corresponds to the RAB$L_ROP field, the ASY option.

## *BLOCK_IO*

Is a switch that controls whether block or record I/O operations are performed. If you set the switch to YES, only block operations are permitted.

If you set the switch to NO, only record operations are allowed for relative and indexed files. However, if you also specify the ACCESS section RECORD_IO attribute, both block and record operations may be performed on sequential files.

This attribute corresponds to the RAB$L_ROP field, the BIO option.

## *BUCKET_IO*

Contains a relative record number or a numeric value representing the virtual block number to be accessed. You use this attribute with records in a relative file or when you want block I/O to be performed.

This attribute corresponds to the RAB$L_BKT field.

### CONTEXT

Contains any user-selected value, up to four bytes in length. CONTEXT is devoted exclusively to your use. VMS RMS does not use this attribute, so you can put any value you want in it. For example, you could use it to communicate with a completion routine in your program.

This attribute corresponds to the RAB$L_CTX field.

### END_OF_FILE

Is a switch indicating that VMS RMS is to position to the end of the file when a Connect operation takes place.

This attribute corresponds to the RAB$L_ROP field, the EOF option.

### FAST_DELETE

Is a switch specifying that pointers from the alternate indexes that allow duplicates are not to be deleted as soon as you delete a record. Instead, the pointers are deleted when you try to access the deleted record, in which case an error message is returned. In other words, the FAST_DELETE attribute prevents the overhead associated with the usual way VMS RMS deletes a record—updating the data level, the primary index, and then the alternate indexes.

This attribute corresponds to the RAB$L_ROP field, the FDL option.

### FILL_BUCKETS

Is a switch specifying that VMS RMS is to load buckets according to the fill size established at file-creation time.

If you do not set the switch, VMS RMS ignores the established bucket fill size, and fills buckets completely.

This attribute corresponds to the RAB$L_ROP field, the LOA option.

### KEY_GREATER_EQUAL

When using an ascending data type, KEY_GREATER_EQUAL is a switch requesting VMS RMS to access the first record in an indexed file containing a value (for the specified key of reference) greater than or equal to the value described by the RAB$L_KBF and RAB$B_KSZ fields. If you are using a descending data type, then KEY_GREATER_EQUAL accesses the first record that contains a value for the specified key of reference less than or equal to the value described by the RAB$L_KBF and RAB$B_KSZ fields.

For more information about the RAB$L_KBF and RAB$B_KSZ fields, refer to the *VMS Record Management Services Manual.*

If you set neither this switch nor the KEY_GREATER_THAN switch, a key equal match is made.

This attribute corresponds to the RAB$L_ROP field, the KGE option.

### KEY_GREATER_THAN

When using an ascending data type, the KEY_GREATER_THAN attribute is a switch requesting VMS RMS to access the first record in an indexed file containing a value (for the specified key of reference) greater than the value described by the RAB$L_KBF and RAB$B_KSZ fields. When using a descending data type, the KEY_GREATER_THAN attribute requests VMS RMS to access the first record that contains a value less than that specified in the RAB$L_KBF and RAB$B_KSZ fields. For more information about the

RAB$L_KBF and RAB$B_KSZ fields, refer to the *VMS Record Management Services Manual.*

If you set neither this switch nor the KEY_GREATER_EQUAL switch, a key equal match is made.

This attribute corresponds to the RAB$L_ROP field, the KGE option.

### KEY_LIMIT

Is a switch indicating that the key value described by the RAB$L_KBF and RAB$B_KSZ fields is to be compared to the value in the record accessed in sequential mode. If you set this switch and the record's key value is greater than the limit key value, then the RMS$_OK_LIM status code is returned.

This attribute corresponds to the RAB$L_ROP field, the LIM option.

### KEY_OF_REFERENCE

Specifies the key or index (such as primary, or first alternate) by which you want to process records in a file. The value 0 indicates the primary key. Values 1 through 254 indicate alternate keys. The default value is 0 (primary key).

KEY_OF_REFERENCE is applicable to indexed files only.

This attribute corresponds to the RAB$B_KRF field.

### LOCATE_MODE

Is a switch specifying that VMS RMS is to return records by supplying a pointer to the data rather than copying the data to the user buffer.

This attribute corresponds to the RAB$L_ROP field, the LOC option.

### LOCK_ON_READ

Is a switch specifying that a record is to be locked for read. Other accessors may read the record while it is locked, but they cannot modify it.

If you specify both the LOCK_ON_READ and the LOCK_ON_WRITE attributes, LOCK_ON_WRITE takes precedence. The NOLOCK attribute takes precedence over both.

This attribute corresponds to the RAB$L_ROP field, the REA option.

### LOCK_ON_WRITE

Is a switch specifying that a record is to be locked for modification. Other accessors may read the record while it is locked.

If you specify both the LOCK_ON_WRITE and the LOCK_ON_READ attributes, LOCK_ON_WRITE takes precedence. The NOLOCK attribute takes precedence over both.

This attribute corresponds to the RAB$L_ROP field, the RLK option.

### MANUAL_UNLOCKING

Is a switch specifying that VMS RMS cannot unlock records automatically. Instead, after a record is locked (by a Get, Find, or Put operation), it must be explicitly unlocked by a Free or Release VMS RMS operation.

The NOLOCK attribute takes precedence over the MANUAL_UNLOCKING attribute.

This attribute corresponds to the RAB$L_ROP field, the ULK option.

### MULTIBLOCK_COUNT

Specifies the number of blocks, in the range 0 to 127, to be allocated to each I/O buffer. MULTIBLOCK_COUNT applies only when accessing a sequential disk file.

The MULTIBLOCK_COUNT attribute optimizes data throughput for sequential operations, and it does not affect the structure of the file. It reduces the number of times you would otherwise have to access the disk for record operations, so execution time is likewise reduced. However, the extra buffering increases memory requirements.

If you do not set this attribute or set it to 0, the process default for the multiblock count is used. If the process default is also 0, RMS uses the system default. If the system default is also 0, then the default size for each I/O buffer is one block. You use the DCL command SET RMS_DEFAULT to set process or system defaults.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$B_MBC field.

### MULTIBUFFER_COUNT

Specifies the number of buffers, in the range 0 to 127, to be allocated at connect time.

If you do not set this attribute or set it to 0, VMS RMS uses the process default for the particular file organization and device type. If the process default is also 0, the system default for the particular file organization and device type applies.

If the system default is likewise 0, one buffer is allocated. However, if you specify either the READ_AHEAD or the WRITE_BEHIND attribute, a minimum of two buffers are allocated. A minimum of two buffers are also allocated for an indexed sequential file or for a process permanent file.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$B_MBF field.

### NOLOCK

Is a switch specifying that the record accessed through a Get or Find RMS operation is not to be locked. The NOLOCK attribute takes precedence over all other attributes controlling record locking, such as MANUAL_UNLOCKING, LOCK_ON_READ, and LOCK_ON_WRITE.

This attribute corresponds to the RAB$L_ROP field, the NLK option.

### NONEXISTENT_RECORD

Is a switch specifying that if a record randomly accessed with a Get or Find RMS operation does not exist (was never inserted into the file or was deleted), the record is to be processed anyway, locking the record cell if necessary.

NONEXISTENT_RECORD does not apply to indexed files.

This attribute corresponds to the RAB$L_ROP field, the NXR option.

### READ_AHEAD

Is a switch used with multiple buffers (see MULTIBUFFER_COUNT), indicating read-ahead operations. It indicates that the system does not have to wait for I/O completion because input and computing can overlap.

In other words, when one buffer is filled, the next record is read into the next buffer while the I/O operation takes place for the first buffer.

If you specify READ_AHEAD when the multibuffer count is 0, two buffers are allocated to allow multibuffering. If you specify two or more buffers, multibuffering is allowed regardless. However, if you specify a buffer count of 1, multibuffering is disabled.

READ_AHEAD is ignored for unit record device I/O. It applies to sequential file processing only.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the RAH option.

### READ_REGARDLESS

Is a switch specifying that a record is to be read even if it is locked. This attribute allows some control over access. In other words, if a record is locked against all access and you request a Find or Get RMS operation, the record is returned anyway.

This attribute corresponds to the RAB$L_ROP field, the RRL option.

### TIMEOUT_ENABLE

Is a switch specifying that the maximum time value, in seconds, is allowed for a record input wait caused by a locked record if the WAIT_FOR_RECORD attribute was specified. This attribute also applies to the time allowed for a character to be received during terminal input. If the timeout period expires, VMS RMS returns an error status.

In addition, TIMEOUT_ENABLE serves a special purpose for mailbox devices. If you specify this attribute with a TIMEOUT_PERIOD of 0, Get and Put RMS operations to mailbox devices use the IO$M_NOW modifier. The operation then completes immediately instead of synchronizing with another cooperating writer or reader of the mailbox. See the *VMS I/O User's Reference Volume* for a further discussion of mailboxes.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the TMO option.

### TIMEOUT_PERIOD

Specifies the maximum number of seconds, in the range 0 through 255, that a Get RMS operation can use. If a Get operation is specified from the terminal and you specify 0, the current contents of the type-ahead buffer are returned.

To use this attribute, you must also specify the TIMEOUT_ENABLE attribute.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$B_TMO field.

### TRUNCATE_ON_PUT

Is a switch specifying that a Put or Write RMS operation can occur at any point in a file, truncating the file at that point. A write operation causes the end-of-file mark to immediately follow the last byte written.

TRUNCATE_ON_PUT can only be used with sequential files.

This attribute corresponds to the RAB$L_ROP field, the TPT option.

### TT_CANCEL_CONTROL_O

Is a switch guaranteeing that terminal output is not discarded if you press CTRL/O.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the CCO option.

### TT_PROMPT

Is a switch indicating that the contents of the prompt buffer are to be used as a prompt on a read from a terminal.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the PMT option.

### TT_PURGE_TYPE_AHEAD

Is a switch eliminating any information that may be in the type-ahead buffer on a read from a terminal.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the PTA option.

### TT_READ_NOECHO

Is a switch indicating that input data is not to be echoed (displayed) on the terminal as it is entered on the keyboard.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the RNE option.

### TT_READ_NOFILTER

Is a switch indicating that CTRL/U, CTRL/R, and DELETE are not to be considered control commands on terminal input, but are to be passed to the user program.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the RNF option.

### TT_UPCASE_INPUT

Is a switch that changes lowercase characters on a read from a terminal to uppercase.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the CVT option.

### UPDATE_IF

Is a switch indicating that if a put operation is specified for a record that already exists in the file, the operation is converted to an update. This attribute is necessary to overwrite (as opposed to update) an existing record in relative and indexed sequential files.

Indexed files using UPDATE_IF must not allow duplicates on the primary key.

This attribute corresponds to the RAB$L_ROP field, the UIF option.

### *WAIT_FOR_RECORD*

Is a switch specifying that VMS RMS should wait for a currently locked record until it becomes available. You can use this attribute with the TIMEOUT_ENABLE and TIMEOUT_PERIOD attributes to limit waiting periods to a specified time.

This attribute corresponds to the RAB$L_ROP field, the WAT option.

### *WRITE_BEHIND*

Is a switch used with multiple buffers (see MULTIBUFFER_COUNT) to indicate write-behind operations. It indicates that the system does not have to wait for I/O completion because computing and output can overlap. In other words, when one buffer is filled, the next record is written into the next buffer while the I/O operation takes place for the first buffer.

If you specify WRITE_BEHIND when the multibuffer count is 0, two buffers are allocated to allow multibuffering. If you specify two or more buffers, multibuffering is allowed regardless. However, if you specify a buffer count of 1, multibuffering is disabled.

WRITE_BEHIND is ignored for unit record device I/O. It applies to sequential file processing only.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the RAB$L_ROP field, the WBH option.

## 1.1.6 DATE Section

The DATE section allows you to specify dates and times for certain file characteristics. The DATE keyword itself takes no values; it serves only to set off this section. Table FDL–5 lists the DATE secondary attributes and their default values.

**Table FDL–5 Default Values for DATE Secondaries**

| Secondary | Default Value |
|---|---|
| BACKUP | Null-string |
| CREATION | Null-string |
| EXPIRATION | Null-string |
| REVISION | Null-string |

In general, you should let the system specify values for the DATE secondaries. The only secondary you can routinely (and safely) set is EXPIRATION.

### *BACKUP*

Is a string that indicates the date when the data file was last backed up. It must be of the following form:

dd-mmm-yyyy hh:mm:ss.cc.

This attribute corresponds to the XAB$Q_BDT field.

### *CREATION*

Is a string that indicates the date and time of the data file's creation. It must be of the following form:

dd-mmm-yyyy hh:mm:ss.cc.

This attribute corresponds to the XAB$Q_CDT field.

### *EXPIRATION*

Is a string that indicates the date and time after which a disk file may be considered for deletion. For magnetic tape files, the EXPIRATION attribute sets the date and time after which you can overwrite the file. It must be of the following form:

dd-mmm-yyyy hh:mm:ss:cc.

This attribute corresponds to the XAB$Q_EDT field.

### *REVISION*

Is a string that indicates the date of the last modification of the data file. It must be of the following form:

dd-mmm-yyyy hh:mm:ss.cc.

This attribute corresponds to the XAB$Q_RDT field.

## 1.1.7 FILE Section

The FILE section allows you to specify file processing and file-related characteristics for your file. The FILE primary keyword takes no values.

FILE section attributes (ALLOCATION, BEST_TRY_CONTIGUOUS, BUCKET_SIZE, CONTIGUOUS, and EXTENSION) have corresponding AREA section attributes. If you specify values for these attributes in the AREA section, they override any values that you specify in the FILE section.

Table FDL–6 lists the FILE secondary attributes and their default values.

**Table FDL–6 Default Values for FILE Secondaries**

| Secondary | Default Value |
|---|---|
| ALLOCATION | 0 |
| BEST_TRY_CONTIGUOUS | NO |
| BUCKET_SIZE | 0 |
| CLUSTER_SIZE | 3 |
| CONTEXT | 0 |
| CONTIGUOUS | NO |
| CREATE_IF | NO |
| DEFAULT_NAME | Null-string |
| DEFERRED_WRITE | NO |
| DELETE_ON_CLOSE | NO |
| DIRECTORY_ENTRY | YES |
| EXTENSION | 0 |

**Table FDL–6 (Cont.) Default Values for FILE Secondaries**

| Secondary | Default Value |
|---|---|
| FILE_MONITORING | NO |
| GLOBAL_BUFFER_COUNT | 0 |
| MAX_RECORD_NUMBER | 0 |
| MAXIMIZE_VERSION | YES |
| MT_BLOCK_SIZE | 0 |
| MT_CLOSE_REWIND | NO |
| MT_CURRENT_POSITION | NO |
| MT_NOT_EOF | NO |
| MT_OPEN_REWIND | NO |
| MT_PROTECTION | Space character |
| NAME | Null-string |
| NON_FILE_STRUCTURED | NO |
| ORGANIZATION | SEQUENTIAL |
| OUTPUT_FILE_PARSE | NO |
| OWNER | System or process default |
| PRINT_ON_CLOSE | NO |
| PROTECTION | System or process default |
| READ_CHECK | NO |
| REVISION | 0 |
| SEQUENTIAL_ONLY | NO |
| SUBMIT_ON_CLOSE | NO |
| SUPERSEDE | NO |
| TEMPORARY | NO |
| TRUNCATE_ON_CLOSE | NO |
| USER_FILE_OPEN | NO |
| WINDOW_SIZE | Volume default |
| WRITECHECK | NO |

## *ALLOCATION*

Sets the number of blocks that you initially allocate for the data file. The value you specify must be an integer in the range 0 to 4,294,967,295. The default is 0, which means that the system does not initially allocate space for the file.

Note that you can override this attribute by specifying the corresponding attribute in the AREA section.

This attribute corresponds to the FAB$L_ALQ field.

### BEST_TRY_CONTIGUOUS

Is a switch that controls whether the file is to be allocated contiguously if there is enough space for it. If you set the switch to YES and if there is enough space for the file, the file is allocated contiguously. If you set the switch to YES and there is not enough space, the file is allocated noncontiguously.

If you set the switch to the default, NO, this attribute is ignored. It is also ignored if no allocation is specified.

Note that you can override this attribute by specifying the corresponding attribute in the AREA section.

This attribute corresponds to the FAB$L_FOP field, the CBT option.

### BUCKET_SIZE

Sets the number of blocks per bucket. Its value must be an integer in the range 0 to 63. The default value is 0, which means that the bucket size computed by VMS RMS is the smallest bucket size capable of holding the largest record. If the file is to be processed by RMS–11, the bucket size is limited to 32 blocks.

If you specify separate areas for the data level and the index levels, then you must define separate bucket sizes for each area. In such a case, this attribute has no meaning because it is overridden when you specify the corresponding attribute in the AREA section.

This attribute corresponds to the FAB$B_BKS field.

### CLUSTER_SIZE

Defines the disk cluster size, which is the number of blocks allocated to a cluster. The system manager or operator determines the disk cluster size when the disk (or volume) is initialized. The disk cluster size can only be set when a volume is initialized.

CLUSTER_SIZE is valid only in the output from the Analyze/RMS_File Utility. ANALYZE/RMS_FILE then returns the actual value of the disk cluster size to EDIT/FDL for use during an Optimize script.

Note that the FDL attribute CLUSTER_SIZE does not have the same meaning as the cluster-size in the RSTS/E operating system.

### CONTEXT

Contains a user-specified value 4 bytes long. This field is intended solely for you to convey user information to a completion routine in your program; VMS RMS never uses it for record management activities.

This attribute corresponds to the FAB$L_CTX field.

### CONTIGUOUS

Is a switch that controls whether the file must be allocated contiguously.

When you set the switch to YES, the file must be allocated contiguously. If there is not enough space for the file's initial allocation, you receive an error.

When you set the switch to the default, NO, the attribute is ignored. It is also ignored if no allocation is specified.

Note that you can override this attribute by specifying the corresponding attribute in the AREA section.

This attribute corresponds to the FAB$L_FOP field, the CTG option.

### CREATE_IF

Is a switch that opens an already existing file. If you set the switch to YES, the file is created if it does not already exist. The alternate success status RMS$_CREATED is then returned to indicate that the file was created, not just opened. The file is input only on an RMS Create service. The CREATE_IF attribute overrides the SUPERSEDE (supersede existing file) attribute.

This attribute corresponds to the FAB$L_FOP field, the CIF option.

### DEFAULT_NAME

Takes a string value that can define portions of the file specification of the data file to be created.

When a utility creates a data file from an FDL file, it first attempts to get the file specification from the call to the utility. If you supply a full file specification with the call to the utility, then the values for the DEFAULT_NAME and NAME attributes are ignored.

If you supply only a partial file specification when you invoke the creating utility, the utility tries to fill in the remainder of the file specification from the value of DEFAULT_NAME. If you do not specify a value for DEFAULT_NAME, the utility uses the VMS RMS defaults.

If you do not supply the utility with a file specification but supply a full file specification with the NAME attribute, the creating utility uses that file specification. If you supply only a partial file specification with the NAME attribute, then the utility uses that portion, and then looks to the DEFAULT_NAME attribute for the rest of the file specification.

If the file specification is still incomplete at that point, the utility uses the VMS RMS defaults to complete the file specification.

For example, if you assign the value WRKD$:.KSM to DEFAULT_NAME, then, unless you specify otherwise, the created data file has the device name WRKD$ and the file type KSM in its file specification.

This attribute corresponds to the FAB$L_DNA and the FAB$B_DNS fields.

### DEFERRED_WRITE

Is a switch that controls whether the writing of modified I/O buffers back to the file is deferred until that buffer is needed for other purposes. This attribute applies only to relative files and indexed files.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the FAB$L_FOP field, the DFW option.

### DELETE_ON_CLOSE

Is a switch that defines the file status after it has been closed. When you set the switch to YES, this attribute specifies that the file is to be deleted after it is closed.

If you set this attribute to YES, you cannot create the file with CREATE/FDL or with FDL$CREATE. They both open and then close the data file, which means that the file will never be around long enough to be used. To create a file that has the DELETE_ON_CLOSE attribute set to YES, you must use the FDL$PARSE routine.

When you set the switch to the default, NO, this attribute is ignored.

This attribute corresponds to the FAB$L_FOP field, the DLT option.

### *DIRECTORY_ENTRY*

Is a switch that defines the file as a temporary one. When you set the switch to the default, YES, it means that the file is created and retained with a directory entry.

When you set this attribute to NO, the file is retained but with no directory entry. To access that file, you must use its file ID.

This attribute corresponds to the FAB$L_FOP field, the TMP option.

### *EXTENSION*

Sets the number of blocks for the default extension value for the file. Each time the file is automatically extended, the specified number of blocks is added. The value for this attribute must be an integer in the range 0 to 65,535. The default is 0, which means the extension size is determined by the system whenever the file must be extended.

Note that you can override this attribute by specifying the corresponding attribute in the AREA section.

This attribute corresponds to the FAB$W_DEQ field.

### *FILE_MONITORING*

Is a switch that corresponds to XAB$_STAT_ENABLE. Enables performance monitoring. The default value is NO.

### *GLOBAL_BUFFER_COUNT*

Specifies the number of global buffers to be allocated for the data file. The value for this attribute must be a number in the range 0 to 32,767. The default value is 0.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the FAB$W_GBC field.

### *MAX_RECORD_NUMBER*

Specifies the maximum number of records that can be placed in a relative file.

The value must be an integer in the range of 0 to 2,147,483,647. The default value is 0, which means that you can place as many records as you want in the relative file, up to the maximum 2,147,483,647.

This attribute corresponds to the FAB$L_MRN field.

### *MAXIMIZE_VERSION*

Is a switch that controls the version number assigned to the file specification of a data file.

If you set the switch to the default, YES, the FDL Facility gives the file specification of the data file the greater of two possible version numbers: either the number that was part of the file specification or a version number that is one higher than the highest existing version number.

When you set the switch to NO, then giving an explicit version number in the file specification that is lower than an existing version results in creating a new data file that has the lower version number. Giving a version number in the file specification that exactly matches an existing one results in an error.

This attribute corresponds to the FAB$L_FOP field, the MXV option.

### MT_BLOCK_SIZE

Sets the number of bytes in a block for magnetic tape files. The value for this attribute is either 0 or an integer in the range 20 to 65,535 for ANSI-formatted tapes or 14 to 65,532 for foreign tapes (tapes that are not in the standard ANSI format used by the VMS operating system and that must be mounted by means of the DCL command MOUNT/FOREIGN). If the default value of 0 is taken, then the block size specified when the tape was mounted is used.

This attribute corresponds to the FAB$W_BLS field.

### MT_CLOSE_REWIND

Is a switch that controls whether a magnetic tape volume is rewound when the file is closed. When you set the switch to YES, the magnetic tape volume is rewound.

When you set the switch to the default value, NO, the magnetic tape volume is not rewound.

This attribute corresponds to the FAB$L_FOP field, the RWC option.

### MT_CURRENT_POSITION

Is a switch that controls the starting position on a magnetic tape where a data file is written. When you set the switch to YES, the data file is written immediately following the current tape file.

In the absence of position specifiers (i.e., MT_CURRENT_POSITION=NO and MT_OPEN_REWIND=NO) the file is created at the logical end of the tape.

This attribute corresponds to the FAB$L_FOP field, the POS option.

### MT_NOT_EOF

Is a switch that prevents positioning to the end of a file when a tape file is opened and the FAB$B_FAC (file access) field of this FAB indicates an RMS Put operation (the ACCESS PUT attribute has been specified).

This attribute corresponds to the FAB$L_FOP field, the NEF option.

### MT_OPEN_REWIND

Is a switch that controls whether a magnetic tape volume is rewound before any Open or Create operation. When the switch is set to YES, the magnetic tape volume is rewound and the current contents of the tape volume or volume set are overwritten.

When you set the switch to the default value, NO, the magnetic tape volume is not rewound. An Open operation begins at the current tape position and writes to the end of the tape. A Create operation rewinds the tape, but then skips over existing data on the tape. This attribute takes precedence over the MT_CURRENT_POSITION attribute (FAB$L_FOP field, the POS option).

This attribute corresponds to the FAB$L_FOP field, the RWO option.

### MT_PROTECTION

Allows you to control access to a magnetic tape file. By default, this attribute takes a space character as its value, which means that access is not controlled. If you assign any character other than the space, then, to access the file, you must specify the /OVERRIDE=ACCESSIBILITY qualifier and option when you initialize or mount the volume.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the XAB$B_MTACC field.

### NAME

Is the file specification of the data file to be created from this FDL file. If you supply a creating utility with a name for the data file, that name overrides the one specified here.

This attribute corresponds to the FAB$L_FNA and the FAB$B_FNS fields.

### NON_FILE_STRUCTURED

Indicates that the volume is to be processed in a manner that is not file-structured.

This attribute is not supported for DECnet operations; an error is returned if you try to use it.

This attribute corresponds to the FAB$L_FOP field, the NFS option.

### ORGANIZATION

Defines the type of file organization. Its value must be one of the following keywords:

- SEQUENTIAL
- RELATIVE
- INDEXED

The default is SEQUENTIAL.

This attribute corresponds to the FAB$B_ORG field.

### OUTPUT_FILE_PARSE

Is a switch specifying that the resultant file specification string, if used, is to provide directory, file name, and file type defaults only.

This attribute corresponds to the FAB$L_FOP field, the OFP option.

### OWNER

Specifies who is to be the owner of the data file. The value that you supply is the user identification code (UIC), in this form:

```
octal-group-number,octal-user-number
```

For example, OWNER [12,322] indicates that the person in group 12 with the user number 322 is the owner of the data file.

This attribute corresponds to the XAB$W_GRP and the XAB$W_MBM fields.

## PRINT_ON_CLOSE

Is a switch that controls whether the data file is to be spooled to the process default print queue when the file is closed. This attribute applies to sequential files only. When you set the switch to YES, the data file is to be spooled to the process default print queue (SYS$PRINT) after the file is closed.

When you set the switch to the default, NO, this attribute is ignored.

If you also set DELETE_ON_CLOSE to YES, the file is deleted after it is printed.

This attribute corresponds to the FAB$L_FOP field, the SPL option.

## PROTECTION

Defines the levels of file protection.

Its value is a string of one of these two forms:

```
SYSTEM=code,OWNER=code,GROUP=code,WORLD=code
SYSTEM:code,OWNER:code,GROUP:code,WORLD:code
```

The code is a protection specification for READ, WRITE, EXECUTE, and DELETE in the form:

```
[R][W][E][D]
```

The default gives the data file the current process default protection. To see what this is, enter the DCL command SHOW PROTECTION.

To deny a specific access right, you omit it from the code. To withhold all access rights from a user classification, omit the classification from the list. For example, the following string gives all access rights to SYSTEM and OWNER, gives only READ access to GROUP, and gives no access rights to WORLD.

```
SYSTEM=RWED,OWNER=RWED,GROUP=R
```

This attribute corresponds to the XAB$W_PRO field.

## READ_CHECK

Is a switch that determines whether transfers from disk volumes are followed by read-compare operations.

When you set the switch to YES, transfers from disk volumes are followed by read-compare operations. This double checking increases the likelihood that the system will catch data errors; however, it also increases disk overhead.

Setting this switch does not permanently mark the file for READ_CHECK; it merely sets an RMS run-time option. Instead, you must use the SET FILE/DATA_CHECK=READ command to mark the file permanently.

When you set the switch to the default, NO, the attribute is ignored.

This attribute corresponds to the FAB$L_FOP field, the RCK option.

### REVISION

Specifies the revision number of the data file. Its value is an integer in the range 0 to 65,535. Unless you want to change the revision number to some specific number, you should leave this value at its default of 0. When REVISION is set to 0, the file's revision number is increased by one every time the file is opened for write access.

This attribute corresponds to the XAB$W_RVN field.

### SEQUENTIAL_ONLY

Is a switch indicating that the file can only be processed sequentially, thus allowing certain processing optimizations. Any attempt to perform random access results in an error.

For DECnet operations, this attribute enables file transfer mode, which is a Data Access Protocol (DAP) feature that allows several records to be transferred in a single network operation. It maximizes throughput for single-direction, sequential-access file transfer.

This attribute corresponds to the FAB$L_FOP field, the SQO option.

### SUBMIT_ON_CLOSE

Is a switch that determines whether the data file is submitted to the process default batch queue (SYS$BATCH) when the file is closed.

When you set the switch to YES, the data file is submitted. This setting makes sense only if the data file is a sequential command file.

When you set the switch to NO, this attribute is ignored.

If you also set DELETE_ON_CLOSE to YES, the file is deleted after the batch job completes.

This attribute corresponds to the FAB$L_FOP field, the SCF option.

### SUPERSEDE

Is a switch that determines whether the existing data file is replaced by a different one of the same name, type, and version. When you set the switch to YES, the existing data file is replaced.

When you set the switch to the default, NO, this attribute is ignored.

If you try to create a new file with the same name, type, and version as an existing file, the old file is deleted if the new one is created successfully.

SUPERSEDE is overridden by the CREATE_IF attribute.

This attribute corresponds to the FAB$L_FOP field, the SUP option.

### TEMPORARY

Is a switch indicating that a temporary file is to be created and deleted when the file is closed. A directory entry is not created for the temporary file.

If you set this attribute to YES, you cannot create the file with CREATE/FDL (or with FDL$CREATE). They both open and then close the data file, which means that the file will never be around long enough to be used. To create a file that has the TEMPORARY attribute set to YES, you must use the FDL$PARSE routine.

This attribute corresponds to the FAB$L_FOP field, the TMD option.

### TRUNCATE_ON_CLOSE

Is a switch that indicates whether any unused space at the end of a sequential file is deallocated when the file is closed. When you set this switch to YES, the space is deallocated. This attribute applies to sequential files only.

When you set this switch to the default, NO, this attribute is ignored.

This attribute corresponds to the FAB$L_FOP field, the TEF option.

### USER_FILE_OPEN

Is a switch indicating that VMS RMS only opens or creates a file; no further RMS operations are performed on the file. If you specify this option, you must also specify the SHARING USER_INTERLOCK attribute unless you have also specified the SHARING PROHIBIT attribute.

This attribute is not supported for DECnet operations; an error is returned if you try to use it.

This attribute corresponds to the FAB$L_FOP field, the UFO option.

### WINDOW_SIZE

Specifies the number of retrieval windows (pointers) you want VMS RMS to maintain in memory for your file. You can specify a numeric value in the range 0 to 127, or 255. A value of 0 indicates that VMS RMS is to use the system default number of retrieval pointers. A value of 255 means to map the entire file, if possible. Values between 128 and 254, inclusive, are reserved for future use.

This attribute is not supported for DECnet operations; it is ignored.

This attribute corresponds to the FAB$B_RTV field.

### WRITE_CHECK

Is a switch that, when set to YES, indicates that transfers to disk are checked by a read-compare operation. However, this operation creates extra system overhead.

Setting this switch does not permanently mark the file for WRITE_CHECK; it sets an RMS run-time option. You must use the SET FILE/DATA_CHECK=WRITE command to mark the file permanently.

When you set this switch to the default, NO, this attribute is ignored.

This attribute corresponds to the FAB$L_FOP field, the WCK option.

### 1.1.8 KEY Section

The KEY primary attribute acts as a header for a section of the FDL file that describes keys. You must specify a separate KEY section for each key of an indexed file. The number of the key being described follows the word KEY (for example, KEY 0, KEY 1, . . . KEY *n*). The KEY value for the primary key must be 0. The KEY value for secondary keys can be numbered from 1 to 254.

The KEY primary attribute corresponds to the XAB$B_REF field.

Table FDL–7 lists the KEY secondary attributes and their default values.

**Table FDL–7 Default Values for KEY Secondaries**

| Secondary | Default Value |
|---|---|
| CHANGES | NO |
| COLLATING_SEQUENCE | None (only present for files with collated keys) |
| DATA_AREA | None |
| DATA_FILL | Same as bucket size |
| DATA_KEY_COMPRESSION | YES |
| DATA_RECORD_COMPRESSION | YES |
| DUPLICATES | NO for primary; YES for alternate |
| INDEX_AREA | None |
| INDEX_COMPRESSION | YES |
| INDEX_FILL | Same as bucket size |
| LENGTH | None |
| LEVEL1_INDEX_AREA | None |
| NAME | Null-string |
| NULL_KEY | NO |
| NULL_VALUE | ASCII null character |
| POSITION | None |
| PROLOG | System or process default |
| SEGn_LENGTH | None |
| SEGn_POSITION | None |
| TYPE | STRING |

#### *CHANGES*

Is a switch that, when set to YES, allows an RMS Update operation to change the value of the key. Such a change is not allowed for the primary key (regardless of this attribute), so the default setting for primary keys is NO. With alternate keys the default setting is also NO, but you can specify YES to allow changes to alternate key values.

This attribute corresponds to the XAB$B_FLG field, the CHG option.

### COLLATING_SEQUENCE

The name of the NCS collating sequence that defines the sorting order of the characters for this key. The value is a string from 1 to 31 characters long. You must supply the value; there is no default.

This attribute corresponds to the XAB$L_COLNAM field.

### DATA_AREA

Identifies the area, in an indexed file with multiple areas, where you place the data records. The value is an integer in the range 0 to 254, which must be the same number as that assigned to the area in an AREA section.

The DATA_AREA, LEVEL1_INDEX_AREA, and INDEX_AREA values are used when the data level and the index levels are placed in separate areas or when each key is placed in its own area.

This attribute corresponds to the XAB$B_DAN field.

### DATA_FILL

Sets the percentage of bytes in each data bucket in the area you want populated initially. If you anticipate that many records will be inserted randomly into the file, this value should be less than 100% of the bytes. The default value is 100%, and the minimum value is 50%. The /FILL_BUCKETS qualifier to the CONVERT command overrides this attribute.

This attribute corresponds to the XAB$W_DFL field. Note that XAB$W_DFL contains a byte count, not a percentage.

### DATA_KEY_COMPRESSION

Is a switch that controls whether leading and trailing repeating characters in the primary key are compressed. The default is YES. For compression to occur, your indexed file must be defined as a Prolog 3 file with the FDL attribute KEY PROLOG. However, KEY PROLOG 3 is the default.

This attribute should be set for DECnet operations.

This attribute corresponds to the XAB$B_FLG field, the KEY_NCMPR option.

### DATA_RECORD_COMPRESSION

Is a switch that controls whether repeating characters are compressed in the data records. The default is YES. For compression to occur, your indexed file must be defined as a Prolog 3 file with the FDL attribute KEY PROLOG. However, KEY PROLOG 3 is the default.

This attribute should be set for DECnet operations.

This attribute corresponds to the XAB$B_FLG field, the DAT_NCMPR option.

### DUPLICATES

Is a switch that controls whether duplicate keys are allowed in the indexed files. For primary keys the default setting is NO, but for alternate keys the default setting is YES. When you set the switch to YES, this attribute specifies that there can be more than one record with the same specific key value.

Duplicate alternate keys can be useful. For example, sorting a customer file on an alternate key of the zip code is a common application and one that requires duplicate keys. Sorting a file on an alternate gender key (male or female) is also an application that requires duplicate keys.

When you set this attribute to NO, duplicate keys are not allowed, and any attempt to write a record where the key would be a duplicate results in an error.

This attribute corresponds to the XAB$B_FLG field, the DUP option.

### *INDEX_AREA*

Identifies the area, in an indexed file with multiple areas, where you place the index levels (other than level 1). The value is an integer in the range 0 to 254, which must be the same number as that assigned to the area in an AREA section.

The INDEX_AREA, DATA_AREA, and LEVEL1_INDEX_AREA values are used when the data level and the index levels are placed in separate areas or when each key is placed in its own area.

This attribute corresponds to the XAB$B_IAN field.

### *INDEX_COMPRESSION*

Is a switch that controls whether leading repeating characters in the index are compressed. The default value is YES. For compression to occur, your indexed file must be defined as a Prolog 3 file with the FDL attribute KEY PROLOG. However, KEY PROLOG 3 is the default.

This attribute should be set for DECnet operations.

This attribute corresponds to the XAB$B_FLG field, the IDX_NCMPR option.

### *INDEX_FILL*

Sets the percentage of bytes in each index level bucket to be populated initially. If you anticipate that many records will be inserted randomly into the file, this value should be less than 100%. The default value is 100% and the minimum value is 50%.

The /FILL_BUCKETS qualifier to the CONVERT command overrides this attribute.

This attribute corresponds to the XAB$W_IFL field, which is a byte count, not a percentage.

### *LENGTH*

Sets the length of the key in bytes. This value, along with the POSITION and TYPE attributes, is used when the key is unsegmented.

This value must be specified; there is no default.

This attribute corresponds to the XAB$B_SIZ0 field.

### *LEVEL1_INDEX_AREA*

Identifies the area where you place the level 1 index in an indexed file with multiple areas. The value is an integer in the range of 0 to 254, which must be the same number as that assigned to the area in an AREA section.

The LEVEL1_INDEX_AREA, DATA_AREA, and INDEX_AREA values are used when the data level and the index levels are placed in separate areas or when each key is placed in its own area.

This attribute corresponds to the XAB$B_LAN field.

### NAME

Assigns a name to a key. The value is a string from 1 to 32 characters long. This is an optional value; the default is no name (blank). The string is padded with ASCII null characters to 32 bytes.

This attribute corresponds to the XAB$L_KNM field.

### NULL_KEY

Controls whether null key values will be allowed in an alternate key field. The value of this attribute is a switch, set to NO by default. When set to NO, it means that all records must contain a valid value for this alternate key.

In some databases, such entries are not desirable; some records will not contain a value for a particular alternate key. By allowing null keys, declaring a null field, and writing the null field as the alternate key for a record, you can include the record in the database.

When you set this attribute to YES, null key values are allowed in the specified alternate index field of a record. Keep in mind that only string keys can have null values.

A null key value is whatever you set it to be with the KEY NULL_VALUE secondary. If a record has the specified null value in its alternate key field, a pathway to that record will not be made in the alternate index structure.

This attribute corresponds to the XAB$B_FLG field, the NUL option.

### NULL_VALUE

Defines the null value that instructs the system not to create an alternate index entry for the record that has the null value in every byte of the key field.

If the alternate key is of the STRING type, you can specify the null value either by specifying the character itself or by specifying an unsigned decimal number denoting the character's ASCII value. To specify the character, enclose it in apostrophes. To specify the decimal ASCII value, just type it with no enclosing characters.

The default is the ASCII null character (which is 0).

This attribute corresponds to the XAB$B_NUL field.

### POSITION

Defines the byte position of the beginning of the key field within the record. The first position is 0. Primary keys work best if they start at byte 0. This attribute, along with the LENGTH and TYPE attributes, is used when the key is unsegmented.

This attribute corresponds to the XAB$W_POS0 field.

### PROLOG

Defines the internal structure level of an indexed file. There are three different structure levels—PROLOG 1, PROLOG 2, and PROLOG 3.

Prolog 3 files accept multiple keys (or alternate keys) and all data types. They also give you the option of compressing your data, indexes, and keys. PROLOG 3 is the default.

On the other hand, Prolog 1 and 2 files do not allow these options. You should not specify Prolog 3 if the primary key is segmented and the segments overlap. If you want to use a Prolog 3 file in this case, consider defining the overlapping segmented key as an alternate key and then choosing a different key to be the primary key.

Note that neither RMS–11 Version 1.8 nor RMS–11 Version 2.0 support Prolog 3 files.

To specify a Prolog 3 file, assign the value 3 to this attribute. To specify a Prolog 1 or 2 file, assign the value 2. There is no perceivable difference between PROLOG 1 and PROLOG 2.

If you do not specify a value for this attribute, then the utility that creates a data file from the FDL file uses the system or process default. To see these default values, enter the DCL command SHOW RMS_DEFAULT.

This attribute is not supported for DECnet operations; the default prolog in effect at the remote node is used.

This attribute corresponds to the XAB$B_PROLOG field.

### *SEGn_LENGTH*

Defines the length of the key segment in bytes. This attribute is used with the SEGn_POSITION attribute when the key is segmented. The *n* is the number of the segment and may be 0 to 7. The first segment in the key must be numbered 0, and each key may have up to eight segments. Segmented keys must be STRING type.

For Prolog 3 files, segments cannot overlap.

This attribute corresponds to any field from XAB$B_SIZ0 to XAB$B_SIZ7.

### *SEGn_POSITION*

Defines the key segment's starting byte position within the record. The first position is 0. Segmented keys must be STRING type.

For Prolog 3 files, segments cannot overlap.

This attribute corresponds to the XAB$W_POS0 (through XAB$W_POS7) field(s).

### *TYPE*

The TYPE attribute specifies the type of the key. The value must be one of the following arguments:

| | |
|---|---|
| BIN2 | An unsigned, 2-byte, binary number in the range 0 to 65,535 ($2^{16}$-1). |
| BIN4 | An unsigned, 4-byte, binary number in the range 0 to 4,294,967,295 ($2^{32}$-1). |
| BIN8 | An unsigned, 8-byte, binary value that ranges from 0 to $2^{64}$-1. |
| COLLATED | A string of ASCII characters. If the key is to be sorted by an NCS collating sequence, then the key type must be declared as COLLATED or as DCOLLATED. The sort order is determined by the collating sequence for that particular key. |

DCOLLATED — A string of ASCII characters. If the key is to be sorted by an NCS collating sequence, then the key type must be declared as COLLATED or as DCOLLATED (descending collated—sort in reverse order according to the collating sequence for that particular key).

DBIN2 — An unsigned, 2-byte, binary value that ranges from 0 to 65,535 ($2^{16}$-1). In an indexed file, records are stored in descending order for this key of reference.

DBIN4 — An unsigned, 4-byte, binary value that ranges from 0 to 4,294,967,295 ($2^{32}$-1). In an indexed file, records are stored in descending order for this key of reference.

DBIN8 — An unsigned, 8-byte, binary value that ranges from 0 to $2^{64}$-1. In an indexed file, records are stored in descending order for this key of reference.

DDECIMAL — A packed-decimal value (that is, a continuous string of between 1 and 16 bytes) accessed in descending sort order in an indexed file. The format of the DDECIMAL type is the same as for DECIMAL, described next (except that DECIMAL is accessed in ascending order).

DECIMAL — A packed-decimal value, which is a continuous string of from 1 to 16 bytes. A DECIMAL value is specified by the address of the first byte of the string and by the number of decimal digits.

Each byte in a DECIMAL value is divided into two 4-bit fields. Each of these fields contains the binary representation of one decimal digit, except for the first 4-bit field in the highest byte, which represents the sign of the DECIMAL value.

Although 4 bits can represent values up to decimal 16 (a hexadecimal 10), values greater than 9 are not allowed in a DECIMAL 4-bit field, except for the sign field.

The byte with address A contains the two beginning digits of the value. The high-order 4-bit field contains either the most significant digit or a leading zero if it is needed to make the sign field appear in the correct 4-bit field.

For example, a DECIMAL value of +123 with address A has a length of 3 (for 3 digits) and requires 2 bytes of storage.

A DECIMAL value of −5237 at address B would have a length of 4 digits. It would need 3 bytes of storage.

DINT2 — A signed, 2-byte integer accessed in descending order in an indexed file. This data type can represent integers between −32,768 and +32,767.

DINT4 — A signed, 4-byte integer accessed in descending order in an indexed file. This data type can represent integers between −2,147,483,648 and +2,147,483,647.

DINT8 — A signed, 8-byte integer accessed in descending order in an indexed file. This data type can represent integers between $-2^{63}$ and $+2^{63}-1$.

DSTRING — A string of ASCII characters accessed in descending sort order in an indexed file. The maximum length of the string is 255 characters.

| | |
|---|---|
| INT2 | A signed, 2-byte integer; this data type can represent integers between −32,768 and +32,767. |
| INT4 | A signed, 4-byte integer; this data type can represent integers between −2,147,483,648 and +2,147,483,647. |
| INT8 | A signed, 8-byte integer; this data type can represent integers between $-2^{63}$ and $+2^{63}-1$. |
| STRING | A string of ASCII characters. The longest length allowed is 255 characters. |

The default key data type is STRING.

This attribute corresponds to the XAB$B_DTP field.

### 1.1.9 NETWORK Section

The NETWORK section sets run-time network access parameters. Table FDL–8 lists the NETWORK secondary attributes and their default values.

**Table FDL–8 Default Values for NETWORK Secondaries**

| Secondary | Default Value |
|---|---|
| BLOCK_COUNT | Varies |
| LINK_CACHE_ENABLE | YES |
| LINK_TIMEOUT | 30 |
| NETWORK_DATA_CHECKING | YES |

#### *BLOCK_COUNT*

Is a number that corresponds to the XABITM item code, XAB$_NET_BLOCK_COUNT, the requested block count. This is the value, in blocks, that the local node uses for buffering messages between itself and the remote node. The value can be 0 to 127. By default, the local node uses the NETWORK BLOCK COUNT value for the process. If that value is 0, then the NETWORK BLOCK COUNT value for the system is used. Use the SHOW RMS command to see what the process and system values are for NETWORK BLOCK COUNT.

#### *LINK_CACHE_ENABLE*

Is a switch that corresponds to the XABITM item code, XAB$_NET_LINK_CACHE_ENABLE. It enables logical link caching.

#### *LINK_TIMEOUT*

Is a number that corresponds to the XABITM item code, XAB$_NET_LINK_TIMEOUT, the logical link timeout in seconds. The value can be from 0 to 65,535.

#### *NETWORK_DATA_CHECKING*

Is a switch that corresponds to the XABITM item code, XAB$_NET_DATA_CRC_ENABLE. Enables Data Access Protocol (DAP) level Cyclic Redundancy Check (CRC).

### 1.1.10 RECORD Section

The RECORD section contains secondary attributes that define records. The RECORD keyword itself takes no value; it serves only to begin this section. Table FDL–9 lists the RECORD secondary attributes and their default values.

**Table FDL–9 Default Values for RECORD Secondaries**

| Secondary | Default Value |
|---|---|
| BLOCK_SPAN | YES |
| CARRIAGE_CONTROL | CARRIAGE_RETURN |
| CONTROL_FIELD | 2 |
| FORMAT | VARIABLE |
| SIZE | No default |

#### *BLOCK_SPAN*

Is a switch that determines whether records can span block boundaries in a sequential file. When the switch is set to YES, the default, records can span block boundaries.

When the switch is set to NO, records cannot span block boundaries; in other words, they cannot be larger than 512 bytes. However, if the records are smaller than 512 bytes, VMS RMS stores as many records as possible in a block until the space remaining is smaller than the next record size. The next record, then, is stored in a new block.

This attribute corresponds to the FAB$B_RAT field, the BLK option.

#### *CARRIAGE_CONTROL*

Must be one of the following keywords:

CARRIAGE_RETURN — Specifies that each record is preceded by a line feed and followed by a carriage return when the record is written to a carriage control device, such as a line printer or a terminal. This is the default.

FORTRAN — Specifies that the first byte (byte 0) of each record contains a FORTRAN (ASA) carriage control character. The following table lists the byte 0 values and the control characters they represent.

| Byte 0 Value | ASCII Character | Meaning |
|---|---|---|
| 0 | null | Null carriage control. Sequence: print buffer contents. |
| 20 | space | Single-space carriage control. Sequence: line feed, print buffer contents, carriage return. |
| 30 | 0 | Double-space carriage control. Sequence: line feed, line feed, print buffer contents, carriage return. |

| Byte 0 Value | ASCII Character | Meaning |
|---|---|---|
| 31 | 1 | Page eject carriage control. Sequence: form feed, print buffer contents, carriage return. |
| 28 | + | Overprint carriage control. Sequence: print buffer contents, carriage return. Allows double printing for emphasis. |
| 24 | $ | Prompt carriage control. Sequence: line feed, print buffer contents. |
| All others | | Same as ASCII space character: single-space carriage control. |

NONE
Specifies that no carriage control is to be provided.

PRINT
Specifies that the carriage control information will come from the fixed control portion of a variable with fixed control (VFC) record. The first byte of the fixed control portion specifies the carriage control to be performed before printing. The second byte specifies the type of carriage control to be performed after printing.

The following tables show the encoding scheme of both bytes.

| Bit 7 | Bits 0-6 | Meaning |
|---|---|---|
| 0 | 0 | No carriage control is specified, that is, NULL. |
| 0 | 1 | Bits 0 through 6 are a count of new lines — a line feed followed by a carriage return. |

| Bit 7 | Bit 6 | Bit 5 | Bit 0-4 | Meaning |
|---|---|---|---|---|
| 1 | 0 | 0 | 0-1F | Output the single ASCII control character specified by the configuration of bits 0 through 4 (7-bit character set). |
| 1 | 1 | 0 | 0-1F | Output the single ASCII control character specified by the configuration of bits 0 through 4. The 5 bits are translated as ASCII characters 128 through 159 (8-bit character set). |
| 1 | 1 | 1 | 0-1F | Reserved. |

This attribute corresponds to the FAB$B_RAT parameter.

## *CONTROL_FIELD*

Specifies the size, in bytes, of the fixed control portion of VFC records. Its value must be a number in the range of 1 to 255. The default is 2.

This attribute corresponds to the FAB$B_FSZ field.

## FORMAT

Sets the record format for the data file. Its value must be one of the following keywords:

| | |
|---|---|
| FIXED | Specifies fixed-length records. |
| STREAM | Specifies that the records are STREAM records; the record is viewed as a continuous stream of bytes, delimited by a special character. This format is compatible with RMS–11 stream files. This is valid for sequential files only. |
| STREAM_CR | Specifies that the records are STREAM records; the record is viewed as a continuous stream of bytes, delimited by a CR character. This is valid for sequential files only. |
| STREAM_LF | Specifies that the records are STREAM records; the record is viewed as a continuous stream of bytes, delimited by an LF character. This is valid for sequential files only. |
| UNDEFINED | Specifies undefined record format, which means that the record is a continuous stream of bytes with no specific terminator. This keyword is valid for sequential files only. |
| VARIABLE | Specifies variable-length records. This is the default setting. |
| VFC | Specifies variable with fixed control records. This is valid for sequential and relative files. |

This attribute corresponds to the FAB$B_RFM field.

## SIZE

Sets the maximum record size in bytes.

When used with fixed-length records, this value is the length of every record in the file.

When used with variable-length records, this value is the longest record that can be placed in the file. With sequential or indexed files, you can specify 0 and the system will not impose a maximum record length. (Note, however, that records in an indexed or relative file cannot cross bucket boundaries.)

When used with relative files, the SIZE attribute is used with the BUCKET_SIZE attribute to set the size of the fixed-length cells.

With VFC records, do not include the fixed control portion of the record in the SIZE calculation; only the data portion is set by this attribute. The RECORD CONTROL_FIELD attribute sets the size of the fixed control portion.

The fixed area is the size in bytes of the fixed-control portion of VFC records. Regular variable-length records have a fixed-control size of 0.

This attribute corresponds to the FAB$W_MRS field.

Table FDL–10 gives the maximum record sizes in bytes for the various record organizations and record formats.

**Table FDL–10 Maximum Record Size for File Organizations and Record Formats**

| File Organization | Record Format | Maximum Record Size |
|---|---|---|
| Sequential | Fixed-length | 32,767 |
| Sequential (disk) | Variable-length | 32,767 |
| Sequential (disk) | VFC | 32,767-FSZ[1] |
| Sequential (disk) | Stream | 32,767 |
| Sequential (disk) | Stream CR | 32,767 |
| Sequential (disk) | Stream LF | 32,767 |
| Sequential (ANSI Tape) | Variable-length | 9,995 |
| Sequential (ANSI Tape) | VFC | 9,995-FSZ[1] |
| Relative | Fixed-length | 32,255 |
| Relative | Variable-length | 32,253 |
| Relative | VFC | 32,253-FSZ[1] |
| Indexed, Prolog 1 or 2 | Fixed-length | 32,234 |
| Indexed, Prolog 1 or 2 | Variable-length | 32,232 |
| Indexed, Prolog 3 | Fixed-length | 32,224 |
| Indexed, Prolog 3 | Variable-length | 32,224 |

[1]The FSZ represents the size of the fixed control area of a record for the variable with fixed control (VFC) record format. The FSZ is equal to the size, in bytes, for the fixed control area of VFC records. The length of the largest record in a sequential file on a disk device with variable or VFC record format is maintained by VMS RMS.

For DECnet operations, the maximum record size is determined by the DCL command SET RMS/NETWORK_BLOCK_COUNT.

### 1.1.11 SHARING Section

The SHARING section allows you to specify whether or not you want to allow multiple readers or writers to access your file at the same time. The SHARING keyword itself takes no values. Table FDL–11 lists the SHARING secondary attributes and their default values.

**Table FDL–11 Default Values for SHARING Secondaries**

| Secondary | Default Value |
|---|---|
| DELETE | None |
| GET | GET if ACCESS GET has also been specified |
| MULTISTREAM | None |
| PROHIBIT | None |
| PUT | None |
| UPDATE | None |
| USER_INTERLOCK | None |

### DELETE
Is a switch allowing other users to delete records from the file.

This attribute corresponds to the FAB$B_SHR field, the DEL option.

### GET
Is a switch allowing other users to read the file (to perform Find or Get RMS services or the equivalent VMS language statement that reads a record). SHARING GET is the default if you have also specified ACCESS GET.

This attribute corresponds to the FAB$B_SHR field, the GET option.

### MULTISTREAM
Is a switch allowing multistream access and is relevant for record operations only. This attribute is not available for sequential files with other than 512-byte, fixed-length records.

This attribute is not supported for DECnet operations; an error is returned if you try to use it.

This attribute corresponds to the FAB$B_SHR field, the MSE option.

### PROHIBIT
Is a switch prohibiting any type of file sharing by other users. If you specify YES, PROHIBIT takes precedence over all other ACCESS secondaries. If you specify the DELETE, PUT, TRUNCATE, or UPDATE attribute in the ACCESS section, the PROHIBIT attribute defaults to YES.

This attribute corresponds to the FAB$B_SHR field, the NIL option.

### PUT
Is a switch allowing other users to write records to the file (to perform Put or Extend RMS services or the equivalent VMS language statement that writes a record or extends the space allocated to a file).

This attribute corresponds to the FAB$B_SHR field, the PUT option.

### UPDATE
Is a switch allowing other users to update records currently existing in the file (to perform Update or Extend RMS services or the equivalent VMS language statement that rewrites a record or extends the space allocated to a file).

This attribute corresponds to the FAB$B_SHR field, the UPD option.

### USER_INTERLOCK
Is a switch allowing one or more users to write to a sequential file or a shared file. Usually this attribute is used for a file that is open for block I/O. You must be responsible for any interlocking required. USER_INTERLOCK is specified with the DELETE, GET, PUT, and UPDATE attributes.

This attribute corresponds to the FAB$B_SHR field, the UPI option.

### 1.1.12 SYSTEM Section

The SYSTEM section consists of system identification information. The SYSTEM primary keyword takes no value. It may be used to help document your FDL file. Table FDL–12 lists the SYSTEM secondary attributes and their default values.

**Table FDL–12 Default Values for SYSTEM Secondaries**

| Secondary | Default Value |
|---|---|
| DEVICE | Null-string |
| SOURCE | VAX/VMS |
| TARGET | VAX/VMS |

#### *DEVICE*

Takes a string value that is used for comment purposes only. The intended use is to name the model of the disk on which the data file will reside, for example, RP06 or RM03.

#### *SOURCE*

Is the name of the operating system you are using to create the FDL file. The value must be one of the following keywords:

- IAS
- RSTS/E
- RSX–11M
- RSX–11M–PLUS
- RT–11
- VAX/VMS

#### *TARGET*

Is the name of the operating system on which the FDL file is to be used. The value must be one of the following keywords:

- IAS
- RSTS/E
- RSX–11M
- RSX–11M–PLUS
- RT–11
- VAX/VMS

### 1.1.13 TITLE and IDENT Attributes

If you use EDIT/FDL to create your FDL file, the utility prompts you for a title during the session. The title is a string that you can place at the beginning of the FDL file. The character string you supply is for comment purposes only. It can be up to 132 characters long, including the TITLE keyword.

When the Edit/FDL and Analyze/RMS_File Utilities create an FDL file, they place a header called the IDENT section after the TITLE in the FDL file. The IDENT section contains the date and time of the creation of the FDL file, and it specifies the name of the utility that created it (either EDIT/FDL or ANALYZE/RMS_FILE).

However, you can also specify the header in the IDENT section. The character string that you supply can be up to 132 characters long, including the IDENT keyword.

## 2 Creating FDL Files

FDL is a powerful tool that can help you easily create the data files for which you have defined specifications. However, you must first create an FDL file containing these specifications. You can create FDL files with one of the following four methods:

- Edit/FDL Utility
- Analyze/RMS_File Utility
- Text editor
- DCL CREATE command

One way to create FDL files easily is with the Edit/FDL Utility (also known as the FDL Editor). You can use the EDIT/FDL command to design FDL files that define commonly needed data files and then to create the data files when they are needed. EDIT/FDL has some special features that simplify the process of creating an FDL file. It recognizes FDL syntax and informs you of syntax errors immediately. It also allows you to model the data file to be created and to change attribute values to find the most efficient design. EDIT/FDL gives files the file type FDL by default.

In addition, the Analyze/RMS_File Utility can create an FDL file from an existing data file. The FDL file can then be used with the EDIT/FDL Optimize script to determine the optimum design of the data file.

You can also use the VMS text editors or the DCL command CREATE to create text files containing FDL specifications. Using the text editors or CREATE is not recommended because you must make sure that you place the primary sections in the correct order and that you give valid values to the attributes. For more information on validity rules, refer to Section 2.1.

The following is an example of a completed FDL file:

```
TITLE   Sequential organization, variable records up to 320 bytes

IDENT   31-DEC-1988 13:08:17    VAX-11 FDL Editor
SYSTEM
        SOURCE                  VAX/VMS
FILE
        ALLOCATION              5050
        BEST_TRY_CONTIGUOUS     yes
        EXTENSION               505
        ORGANIZATION            sequential

RECORD
        BLOCK_SPAN              yes
        CARRIAGE_CONTROL        carriage_return
        FORMAT                  variable
        SIZE                    320
```

## 2.1 Validity Rules

The Edit/FDL and Analyze/RMS_File Utilities place the attributes in their correct format and order automatically. If you use the CREATE command or a text editor to create an FDL file, you must observe the following validity rules:

- The primary sections must appear in the order listed in Section 1.1. If you have two or more AREA primary sections, they must follow one another in numerical order (for example, AREA 1, AREA 2, . . . ,AREA *n*).
- If you have two or more KEY primary sections, they too must follow one another in numerical order (for example, KEY 0, KEY 1, . . . ,KEY *n*).
- Within a KEY primary, any SEGn secondaries should follow one another in numerical order; the SEGn numbers must be "dense," not "sparse." For example, if you use SEG3 to label a key segment, segments SEG0, SEG1, and SEG2 must also exist.
- Each source line can contain exactly one primary or secondary attribute along with its associated value. Each source line may have no more than 132 characters.
- To begin a comment, use the exclamation point. Comments begin at the exclamation point and continue to the end of the line.
- EDIT/FDL ignores leading or trailing blanks or tabs.
- FDL string values are terminated by the comment character (!) or the statement terminator (;). Strings must be enclosed in quotation marks.
- You may truncate keywords, but take care to avoid ambiguities. The Edit/FDL and Analyze/RMS_File Utilities always write out the entire keyword.

# 3 Creating Data Files with RMS Utilities, Routines, and FDL Files

Once you have created an FDL file, it can be used by the RMS utilities and callable utility routines to format data files according to your specifications. Specifically, the RMS utilities CREATE/FDL and CONVERT, as well as the CONVERT and FDL callable utility routines, use FDL files. In addition, EDIT/FDL can use an existing FDL file as an input to the Optimize script.

CREATE/FDL uses the specifications in an existing FDL file to create a new, empty data file. You can either supply CREATE/FDL with the file specification of the new data file, or CREATE/FDL can use the specification given in the FDL file itself.

The Convert Utility, on the other hand, uses the specifications in an FDL file to create an output data file and to load it with records from an input file or files.

Like the Convert Utility, the Convert routines (CONV$CONVERT, CONV$PASS_FILES, and CONV$PASS_OPTIONS) use the specifications in FDL files to create output data files from within a program.

These data files can use the full set of VMS RMS creation-time options. They can be used by all the native VMS high-level languages. This capability gives the high-level language user a tool for creating efficient data files that use a minimum amount of system resources. VAX MACRO and BLISS-32 programs can also use the data files.

The FDL routines (FDL$CREATE, FDL$GENERATE, and FDL$PARSE) also use FDL files. FDL$CREATE invokes the functions of the Create/FDL utility to create a file from an FDL specification and then to close the file. FDL$GENERATE produces an FDL specification from the RMS control blocks your program supplies and then writes it to either an FDL file or a character string. FDL$PARSE parses an FDL specification, allocates RMS control blocks (FABs, RABs, or XABs), and then fills in the relevant fields.

# FDL Usage Summary

File Definition Language (FDL) is a special-purpose language used to write specifications for data files. These specifications are written in text files called FDL files; they are then used by the VMS RMS utilities and library routines to create the actual data files.

One of the RMS utilities, EDIT/FDL, can help you create these FDL files. EDIT/FDL was developed especially to manipulate FDL files. It has some special features designed to simplify the process of creating an FDL file and should be used in most cases.

Another RMS utility, CREATE/FDL, uses the specifications in an existing FDL file to create a new, empty data file.

## FORMAT

**CREATE/FDL=***fdl-filespec [filespec]*

## PARAMETERS

### *fdl-filespec*

Specifies the FDL file from which to create the data file. The default file type is FDL.

### *filespec*

Specifies an optional file specification for the created file. If you specify a complete file specification, it overrides any contained in the FDL file. The default file type is FDL.

## FORMAT

**EDIT/FDL** *fdl-filespec*

## PARAMETER

### *fdl-filespec*

Specifies the FDL file to be created, modified, or optimized during this session. The default file type is FDL.

## usage summary

To invoke the Create/FDL Utility, enter the CREATE/FDL command at the DCL command level. CREATE/FDL produces the empty data file specified by the command line or by the FDL file. To exit the Create/FDL Utility, let it run to successful completion.

To invoke the Edit/FDL Utility, enter the EDIT/FDL command at the DCL command level. EDIT/FDL produces a new version of the input file unless the /OUTPUT qualifier is used to direct the ouput to a different file. To exit the Edit/FDL Utility, enter either the EXIT command or the QUIT command. (Pressing CTRL/Z has the same effect as entering the EXIT command, and CTRL/C has the same effect as the QUIT command.)

**Note: When you enter the EDIT/FDL command, the system refers to a predefined logical name, EDF. If you create your own logical name for EDF, the system cannot execute the EDIT/FDL command correctly. Make sure you do not use EDF for logical names that you create.**

## CREATE/FDL QUALIFIER

The Create/FDL Utility has only one command qualifier — the /LOG qualifier. It does not affect the execution of the utility; it only produces an informational message.

# /LOG

Controls whether the Create/FDL Utility displays the file specification of the data file it has created. By default, the utility does not display the file specification.

## FORMAT

**/LOG**
**/NOLOG**

## PARAMETERS

*None.*

## EXAMPLES

**1**

```
$ CREATE/FDL=INVENTORY/LOG DISK$:[COMPANY.ORDERS]PARTS.DAT
%FDL-I-CREATED, DISK$:[COMPANY.ORDERS]PARTS.DAT;1 CREATED
```

This command produces the empty output file PARTS.DAT from the specifications in the FDL file INVENTORY.FDL. In addition, CREATE/FDL returns the message stating that the file was indeed created.

**2**

```
$ CREATE/FDL=INVENTORY/NOLOG PARTS.DAT
```

This command produces the empty output file PARTS.DAT from the specifications in the FDL file INVENTORY.FDL. No informational message is returned.

## EDIT/FDL QUALIFIERS

The DCL command EDIT/FDL begins an interactive session during which you can create or modify an FDL file. You can supply the editor with file design decisions and it will supply values for the FDL attributes; or you can assign values to the attributes yourself.

# /ANALYSIS

Indicates that an FDL file (which must have been generated by the Analyze/RMS_File Utility) is to be used in the Optimize script.

## FORMAT

**/ANALYSIS=*fdl-filespec***

## QUALIFIER VALUE

***fdl-filespec***
Specifies the particular FDL file (which must have been generated by the Analyze/RMS_File Utility) to be used in the Optimize script. The default is a null specification.

## EXAMPLE

```
$ EDIT/FDL/ANALYSIS=Q1_SALES Q2_SALES
```

This command begins an interactive session in which the analysis information in the file Q1_SALES.FDL is used to optimize and then create the output file Q2_SALES.FDL.

# /CREATE

Allows you to create an output file without an existing input file.

## FORMAT /CREATE

## PARAMETERS *None.*

## DESCRIPTION

With the /CREATE qualifier, you can create an output file without receiving a message from EDIT/FDL stating that the file is to be created. EDIT/FDL does not even try to open the specified file for input; when you use the /CREATE qualifier, EDIT/FDL knows the file does not exist (or that you want EDIT/FDL to ignore it).

You can select the Design or the Add Key scripts only when your input file does not already exist.

## EXAMPLE

```
$ EDIT/FDL/CREATE SALES_DATA
```

Begins a session in which SALES_DATA.FDL is created. EDIT/FDL does not issue the informational message stating that the new file SALES_DATA.FDL will be created.

# /DISPLAY

Specifies the type of graph you want displayed.

## FORMAT

**/DISPLAY=***graph-option*

## QUALIFIER VALUE

***graph-option***

Specifies the type of graph you want displayed. Valid graph options are as follows:

| | |
|---|---|
| LINE | Plots bucket size against index depth |
| FILL | Plots bucket size by the percentage of load fill by index depth |
| KEY | Plots bucket size by key length by index depth |
| RECORD | Plots bucket size by record size by index depth |
| INIT | Plots bucket size by initial load record count by index depth |
| ADD | Plots bucket size by additional record count by index depth |

The default is LINE.

## EXAMPLE

```
$ EDIT/FDL/DISPLAY=KEY TEMP_DATA
```

This command begins an interactive session in which the default value for the type of graph to be displayed has been changed from LINE to KEY. TEMP_DATA is the name of the FDL file to be created.

# /EMPHASIS

Allows you to choose between smaller buffers and flatter files. You can use this qualifier with the /NOINTERACTIVE qualifier if you want EDIT/FDL to be executed without an interactive terminal dialogue.

## FORMAT

**/EMPHASIS=***tuning-bias*

## QUALIFIER VALUE

***tuning-bias***

Represents how you want to weight the default bucket size for your file. There are two valid options:

| | |
|---|---|
| FLATTER_FILES | Generally increases bucket size. The bucket size, in turn, controls the number of levels in the index structure. If a larger bucket size eliminates one level, then you should use this option. At some point, however, the benefit of having fewer levels will be offset by the cost of scanning through the larger buckets. |
| SMALLER_BUFFERS | Generally decreases the amount of memory you have to use. |

The default is FLATTER_FILES. It should be used unless excessive paging or RMS CPU time occurs because of oversized buffers. However, if your system has little extra memory or if you are not sure which tuning-bias will improve the performance of your program, try tuning your file using SMALLER_BUFFERS and then FLATTER_FILES.

## EXAMPLE

```
$ EDIT/FDL/EMPHASIS=SMALLER_BUFFERS TEMP_DATA
```

This command begins an interactive session in which the default value for the bucket size emphasis has been changed from FLATTER_FILES to SMALLER_BUFFERS. TEMP_DATA is the name of the FDL file to be created.

# /GRANULARITY

Allows you to divide an indexed file into a specified number of areas. Use this qualifier with the /NOINTERACTIVE qualifier if you want EDIT/FDL to be executed without an interactive terminal dialogue.

## FORMAT

/GRANULARITY=*n*

## QUALIFIER VALUE

***n***

Indicates the number of areas into which you want to divide your indexed file. The default is three areas, as shown in the following table.

| Area | Contents |
|---|---|
| 0 | KEY 0 data |
| 1 | KEY 0 index |
| 2 | All other indexes |

## EXAMPLE

```
$ EDIT/FDL/GRANULARITY=1 TEMP_DATA
```

This command begins an interactive session in which the default value for the number of areas in an indexed file has been changed from three areas to the one area. TEMP_DATA is the name of the FDL file to be created.

# /NOINTERACTIVE

Causes EDIT/FDL to execute the Optimize script without a terminal dialogue.

## FORMAT

/NOINTERACTIVE

## PARAMETERS

*None.*

## DESCRIPTION

The /NOINTERACTIVE qualifier allows you to optimize an existing FDL file with EDIT/FDL but without an interactive terminal dialogue. You must have previously entered the ANALYZE/RMS_FILE/FDL command, specifying your existing RMS data file as the target file. EDIT/FDL then uses the data from the analysis FDL file while the Optimize script proceeds noninteractively. If data is missing, EDIT/FDL uses the defaults. However, if certain critical data items cannot be found in the analysis file, EDIT/FDL terminates without producing an output file.

## EXAMPLE

```
$ EDIT/FDL/ANALYSIS=TEMP_DATA/NOINTERACTIVE TEMP_DATA
```

This command begins a noninteractive session in which the FDL file TEMP_DATA;2 is created from the analysis FDL file TEMP.DATA;1.

# /NUMBER_KEYS

Allows you to specify the number of keys in your indexed file.

## FORMAT

**/NUMBER_KEYS=*n***

## QUALIFIER VALUE

***n***
Indicates how many keys you want to define for your indexed file. You can define up to 255 keys. The default is one key.

## EXAMPLE

```
$ EDIT/FDL/NUMBER_KEYS=3 TEMP_DATA
```

This command begins an interactive session in which the default value for the number of keys in an indexed file is changed from one key to three keys. TEMP_DATA is the name of the FDL file to be created.

# /OUTPUT

Specifies the FDL file in which to place the definition from the current session.

## FORMAT

**/OUTPUT=***fdl-filespec*

## QUALIFIER VALUE

***fdl-filespec***
Specifies the output FDL file.

## DESCRIPTION

If you omit the /OUTPUT qualifier, then the output FDL file will have the same name and file type as the input file, with a version number that is one higher than the highest existing version of the file.

The default file type is FDL.

## EXAMPLE

```
$ EDIT/FDL/OUTPUT=NEWINDEX INDEX
```

Begins a session in which the contents of INDEX.FDL are read into the FDL editor and can then be modified. NEWINDEX.FDL is created; INDEX.FDL is not changed.

# /PROMPTING

Specifies the level of prompting to be used during the terminal session.

## FORMAT

**/PROMPTING=***prompt-option*

## QUALIFIER VALUE

***prompt-option***

Specifies the level of menu prompting to be used during the terminal session. Valid prompt options are as follows:

| | |
|---|---|
| BRIEF | Selects a terse level of prompting |
| FULL | Provides more information about each menu question |

By default, EDIT/FDL chooses either BRIEF or FULL, depending on the terminal type and the line speed. High-speed CRT terminals are set to FULL; nonscope terminals and terminals operating at less than 2400 baud are set to BRIEF.

If EDIT/FDL has to repeat a question, it repeats the FULL version of the question, with a BRIEF form of the HELP text. You can also type a question mark (?) for help on a particular question.

The extra line of HELP text is not given for menu questions, however.

## EXAMPLE

```
$ EDIT/FDL/PROMPTING=BRIEF TEMP_DATA
```

This command begins an interactive session in which the value of the prompting level for the EDIT/FDL menus is set to BRIEF.

# /RESPONSES

Allows you to select how you want to respond to script questions.

## FORMAT

**/RESPONSES=***response-option*

## QUALIFIER VALUE

***response-option***

Specifies the type of script response you want to use. The two valid options are as follows:

| | |
|---|---|
| AUTOMATIC | Indicates that you want all script default responses to be used automatically. This option speeds the progress of the question and answer session. Once you have entered the design phase, you can modify most of the answers you took by default. |
| MANUAL | Indicates that you want to provide all script responses. No default responses are automatically used. |

If you use the SET RESPONSES function, AUTOMATIC is the default. For EDIT/FDL, however, MANUAL is the default.

## EXAMPLE

```
$ EDIT/FDL/RESPONSES=MANUAL TEMP_DATA
```

This command begins an interactive session in which the type of script response is changed from AUTOMATIC (the default) to MANUAL.

# /SCRIPT

Controls whether EDIT/FDL begins the session by asking a logically grouped sequence of questions to aid you in creating the FDL file.

## FORMAT

**/SCRIPT=*script-title***

## QUALIFIER VALUE

***script-title***

Identifies the seven valid script titles. The valid options are as follows:

| | |
|---|---|
| ADD_KEY | Allows you to model or add to the attributes of a new index. |
| DELETE_KEY | Allows you to remove attributes from the highest index of your file. |
| INDEXED | Begins a dialogue in which you are prompted for information about the indexed data file to be created from the FDL file. EDIT/FDL supplies values for certain attributes. |
| OPTIMIZE | Requires that you use the analysis information from an FDL file that was created with the Analyze/RMS_File Utility. The FDL file itself is one of the inputs to the Edit/FDL Utility. In other words, you may tune the parameters of all your indexes using the file statistics from ANALYZE/RMS_FILE. |
| RELATIVE | Begins a dialogue in which you are prompted for information about the relative data file to be created from the FDL file. EDIT/FDL supplies values for certain attributes. |
| SEQUENTIAL | Begins a dialogue in which you are prompted for information about the sequential data file to be created from the FDL file. EDIT/FDL supplies values for certain attributes. |
| TOUCHUP | Begins a dialogue in which you are prompted for information about the changes you want to make to an existing index. |

## DESCRIPTION

The default is not to invoke a script automatically. Note that, if you specify /NOSCRIPT, you can still use the scripts by entering the INVOKE command in response to the main editor function prompt.

## EXAMPLE

```
$ EDIT/FDL/SCRIPT=INDEXED TEMP_DATA
```

This command begins an interactive session in which both the main menu and the script menu are bypassed. Instead, the Indexed script is generated immediately.

## EDIT/FDL COMMANDS

The EDIT/FDL commands are used during the interactive session only. EDIT/FDL prompts for one of the following commands at the start of your interactive session:

ADD
DELETE
EXIT
HELP
INVOKE
MODIFY
QUIT
SET
VIEW

However, because EDIT/FDL is not command oriented but menu oriented, the prompt may change during the interactive session to fit the needs of the menu questions. In general, the prompt consists of a short question, the type of required value or the range of acceptable values (in parentheses), and the default answer (in brackets), as follows:

```
question        (keyword or range)[default] : answer
```

In addition, some prompts consist of a short question, a list or a range of acceptable values (either in parentheses or in a table), the required type of the value (in parentheses), and the default answer (in brackets), as follows:

```
(list of values)
question        (keyword or range)[default] : answer
```

If no default is allowed, you see the symbol [-], in which case you must supply an answer.

# ADD

Allows you to add one or more lines to the FDL file.

## FORMAT ADD

## PARAMETERS *None.*

## QUALIFIERS *None.*

## EXAMPLE

```
Main Editor Function     (Keyword)[Help] : ADD
```

This command allows you to add lines to your existing FDL file. When you enter the ADD command, EDIT/FDL prompts you with another menu:

```
                    Legal Primary Attributes

ACCESS     attributes set the run-time access mode of the file
ACL        entries specify the Access-Control-List of the file
AREA x     attributes define the characteristics of file area x
CONNECT    attributes set various RMS run-time options
DATE       attributes set the date parameters of the file
FILE       attributes affect the entire RMS data file
JOURNAL    attributes set the journaling parameters of the file
KEY y      attributes define the characteristics of key y
RECORD     attributes set the non-key aspects of each record
SHARING    attributes set the run-time sharing mode of the file
SYSTEM     attributes document operating system-specific items
TITLE      is the header line for the FDL file

Enter desired primary     (Keyword)[FILE] :
```

After you type the name of the primary attribute, EDIT/FDL provides another menu showing all the secondary attributes for that primary and asks which secondary's value you want to change.

# DELETE

Allows you to delete one or more lines from the FDL file.

## FORMAT DELETE

## PARAMETERS *None.*

## QUALIFIERS *None.*

## EXAMPLE

```
Main Editor Function    (Keyword)[Help] : DELETE
```

This command allows you to delete lines from your existing FDL file. When you enter the DELETE command, EDIT/FDL prompts you with a menu displaying the current primary attributes of your FDL file. After you type the name of a primary attribute, EDIT/FDL prompts you with another menu displaying the current secondary attributes for that primary and asks which secondary's value you want to change.

# EXIT

Ends the EDIT/FDL session. The EXIT command causes the new FDL file to be created. This command is equivalent to pressing CTRL/Z.

## FORMAT

**EXIT**

## PARAMETERS

*None.*

## QUALIFIERS

*None.*

## EXAMPLE

```
Main Editor Function      (Keyword)[Help] : EXIT
```

This command allows you to leave EDIT/FDL after creating or modifying your FDL file. It displays the file specification of the FDL file it has created or modified and then returns you to DCL command level.

# HELP

Invokes a help session about the EDIT/FDL commands and the File Definition Language on the screen.

## FORMAT HELP

## PARAMETERS *None.*

## QUALIFIERS *None.*

## EXAMPLE

```
Main Editor Function     (Keyword)[Help] : HELP


        Information available:

        Abstract   Add    Delete   Exit   Help   Invoke   Modify
        Operation  Quit   Set      View

        Topic?
```

This command allows you to request information about EDIT/FDL while you are editing your FDL file. It displays a menu of the various topics about which you can request help.

# INVOKE

Prompts for your choice of scripts and initiates your choice. The scripts guide you through the design and optimization of a data file.

## FORMAT

INVOKE

## PARAMETERS

*None.*

## QUALIFIERS

*None.*

## EXAMPLE

```
Main Editor Function     (Keyword)[Help] : INVOKE


                              Script Title Selection

    Add_key      modeling and addition of a new index's parameters
    Delete_key   removal of the highest index's parameters
    Indexed      modeling of parameters for an entire Indexed file
    Optimize     tuning of all indices' parameters using file
                 statistics
    Relative     selection of parameters for a Relative file
    Sequential   selection of parameters for a Sequential file
    Touchup      remodeling of parameters for a particular index

    Editing Script Title     (Keyword)[-] :
```

This command allows you to select which script you want to help you design your FDL file. After you enter the INVOKE command, EDIT/FDL prompts you with another menu displaying the possible script choices.

# MODIFY

Allows you to change an existing line in the FDL definition.

## FORMAT MODIFY

## PARAMETERS *None.*

## QUALIFIERS *None.*

## EXAMPLE

```
Main Editor Function    (Keyword)[Help] : MODIFY
```

This command allows you to modify lines in your existing FDL file. When you enter the MODIFY command, EDIT/FDL prompts you with a menu displaying the current primary attributes of your FDL file. After you enter the name of a primary attribute, EDIT/FDL prompts you with another menu displaying the current secondary attributes for that primary and asks which secondary's value you want to change.

# QUIT

Causes an abrupt end to the EDIT/FDL session. The new FDL file is not created. The QUIT command is equivalent to pressing CTRL/C.

## FORMAT

QUIT

## PARAMETERS

*None.*

## QUALIFIERS

*None.*

## EXAMPLE

```
Main Editor Function      (Keyword)[Help] : QUIT
```

This command returns you to the DCL command level without creating or modifying an FDL file.

# SET

Allows you to establish defaults or to select any of the FDL editor characteristics you forgot to specify on the command line.

## FORMAT SET

## PARAMETERS *None.*

## QUALIFIERS *None.*

## EXAMPLE

```
Main Editor Function     (Keyword)[Help] : SET

                              FDL Editor SET Function

    ANALYSIS      filespec of FDL Analysis file
    DISPLAY       type of graph to display
    EMPHASIS      of default bucketsize calculations
    GRANULARITY   number of areas in Indexed files
    NUMBER_KEYS   number of keys in Indexed files
    OUTPUT        filespec of FDL output file
    PROMPTING     Full of Brief prompting of menus
    RESPONSES     usage of default reponses in scripts

    Editor characteristic to set     (Keyword)[-] :
```

This command allows you to establish defaults and to reduce the number of questions you are asked by the scripts. After you enter the SET command, EDIT/FDL displays a menu of FDL editor characteristics.

# VIEW

Displays the attributes contained in the current FDL definition.

## FORMAT VIEW

## PARAMETERS *None.*

## QUALIFIERS *None.*

## EXAMPLE

```
Main Editor Function      (Keyword)[Help] : VIEW
```

This command displays your current FDL file a screen at a time.

## FDL EXAMPLES

**1** `$ EDIT/FDL INDEX`

This command begins an interactive session that will modify an FDL file named INDEX.FDL.

**2** `$ EDIT/FDL/ANALYSIS=INDEXFILE/SCRIPT=OPTIMIZE MAKEINDEX`

This command uses the analysis information in INDEXFILE.FDL to create a more efficient MAKEINDEX.FDL. The sequence of events is as follows:

1. The FDL file MAKEINDEX.FDL is created by EDIT/FDL.
2. INDEXFILE.DAT is created by the CREATE/FDL=MAKEINDEX command.
3. INDEXFILE.DAT is used in applications.
4. INDEXFILE.FDL is created with the ANALYZE/RMS_FILE/FDL command.
5. INDEXFILE.FDL is used to optimize MAKEINDEX.FDL.
6. Enter the following command:

```
$ CONVERT/FDL=MAKEINDEX INDEXFILE.DAT INDEXFILE.DAT
```

**3**
```
$ EDIT/FDL/NOINT/A=INVENTORY/G=4
File: SALES
$
```

This command creates the output FDL file SALES from the analysis FDL file INVENTORY without an interactive terminal dialogue. In addition, EDIT/FDL optimizes the input file, changing the granularity factor to four areas and the number of keys to two. Otherwise, all the defaults supplied by EDIT/FDL are used.

# Index

## A



## B



## C

# D



# E



# F

# G



# H



# I

# K



# L



# M



# N

# O



# P



# Q



# R

# S



# T



# U

# V



# W



# X



# Y

# Reader's Comments

**VMS File Definition Language Facility Manual**
AA–LA81A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

| **I rate this manual's:** | Excellent | Good | Fair | Poor |
|---|---|---|---|---|
| Accuracy (software works as manual says) | ☐ | ☐ | ☐ | ☐ |
| Completeness (enough information) | ☐ | ☐ | ☐ | ☐ |
| Clarity (easy to understand) | ☐ | ☐ | ☐ | ☐ |
| Organization (structure of subject matter) | ☐ | ☐ | ☐ | ☐ |
| Figures (useful) | ☐ | ☐ | ☐ | ☐ |
| Examples (useful) | ☐ | ☐ | ☐ | ☐ |
| Index (ability to find topic) | ☐ | ☐ | ☐ | ☐ |
| Page layout (easy to find information) | ☐ | ☐ | ☐ | ☐ |

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

| Page | Description |
|---|---|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line

# Reader's Comments

**VMS File Definition Language Facility Manual**
AA–LA81A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

| **I rate this manual's:** | Excellent | Good | Fair | Poor |
|---|---|---|---|---|
| Accuracy (software works as manual says) | ☐ | ☐ | ☐ | ☐ |
| Completeness (enough information) | ☐ | ☐ | ☐ | ☐ |
| Clarity (easy to understand) | ☐ | ☐ | ☐ | ☐ |
| Organization (structure of subject matter) | ☐ | ☐ | ☐ | ☐ |
| Figures (useful) | ☐ | ☐ | ☐ | ☐ |
| Examples (useful) | ☐ | ☐ | ☐ | ☐ |
| Index (ability to find topic) | ☐ | ☐ | ☐ | ☐ |
| Page layout (easy to find information) | ☐ | ☐ | ☐ | ☐ |

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

| Page | Description |
|---|---|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line