File Utility Program (FUP) Reference Manual

Abstract

This manual describes the command syntax and error messages for the File Utility Program (FUP).

Product Version

T6553 D45, G08, and H01

Supported Release Version Updates (RVUs)

This publication supports D46.00 and all subsequent D-series RVUs, G06.23 and all subsequent G-series RVUs, and H06.03 and all subsequent H-series RVUs until otherwise indicated by its replacement publication.

Part Number	Published
523323-010	November 2006

Document History

Part Number	Product Version	Published
523323-004	FUP D45 and G08	April 2004
523323-007	FUP D45, G08, and H01	July 2005
523323-008	FUP D45, G08, and H01	February 2006
523323-009	FUP D45, G08, and H01	July 2006
523323-010	FUP D45, G08, and H01	November 2006

File Utility Program (FUP) Reference Manual

Glossary | Index | Examples | Tables

1. FUP Overview (continued)

2. FUP Commands

2. FUP Commands (continued)

2. FUP Commands (continued)

2. FUP Commands (continued)

2. FUP Commands (continued)

2. FUP Commands (continued)

3. FUP Messages

A. DEFINE Tables

B. FUP Command Summary

C. FUP Command Syntax Summary

Glossary

Index

Examples

Tables

Tables (continued)

What's New in This Manual

Manual Information

Abstract

This manual describes the command syntax and error messages for the File Utility Program (FUP).

Product Version

T6553 D45, G08, and H01

Supported Release Version Updates (RVUs)

This publication supports D46.00 and all subsequent D-series RVUs, G06.23 and all subsequent G-series RVUs, and H06.03 and all subsequent H-series RVUs until otherwise indicated by its replacement publication.

Part Number	Published
523323-010	November 2006

Document History

Part Number	Product Version	Published
523323-004	FUP D45 and G08	April 2004
523323-007	FUP D45, G08, and H01	July 2005
523323-008	FUP D45, G08, and H01	February 2006
523323-009	FUP D45, G08, and H01	July 2006
523323-010	FUP D45, G08, and H01	November 2006

New and Changed Information

Changes in the H06.08 manual:

- Added the open state, B (Broken) and it's description under INFO Listing Format on page 2-85.
- Updated the note on page 2-87 about the appearance of plus sign (+) to include support for H-series RVUs.
- Added a new file code 547 on page 2-92.
- Consolidated the file codes, 550-565 on page 2-92, in one row with the definition SQL/MX files.
- Updated the VERSION information on page 2-99 and added a reference to a separate manual for more information on SQL/MX versioning.

- Added a new note on pages 2-183 and 2-186 about passing filenames in uppercase while using SPI interface.

Changes in the G06.29 Manual

- Added a new note on page 2-87 about the appearance of plus sign (+) when an OSS file is protected by POSIX access control list (ACL) .
- Added an example of FUP short INFO for OSS files with POSIX ACL on page 2-97.
- Added a note on page 2-149 about the RELOAD operation on ORSERV object using the SHARE option.

Changes in the H06.05 Manual

- Documented that FUP supports fully-qualified SQL/MX ANSI names for the commands — INFO, LISTLOCKS, LISTOPENS, and RELOAD on page 1-1.
- Documented that FUP has a limitation that the command line cannot exceed 132 characters and provided tips on how to use longer ANSI names on page 1-1.
- Added a note on using two separate commands for ANSI names and Guardian names on page 1-1.
- Added missing file codes and the corresponding definitions in Table 2-2 on page 2-87.
- Added the TYPE variable and its description to the key column example on page 2-100.
- Added a note on page 2-100 that the key column TYPE and LENGHT is not displayed for SQL/MX objects .
- Updated these commands to document the FUP support for SQL/MX ANSI names:
  - INFO on page 2-80
  - LISTLOCKS on page 2-116
  - LISTOPENS on page 2-122
  - RELOAD on page 2-146
- Removed the sentence about specifying the same *pri-extent-size* and *sec-extent-size* for all partitions under pri-extent-size , sec-extent-size on page 2-172.
- Added new error messages on page 3-33 through 3-37 that might occur when using SQL/MX ANSI names for the commands - INFO, LISTLOCKS, LISTOPENS, and RELOAD.
- Updated the syntax diagrams for:

- INFO command on page C-6
- LISTLOCKS command on page C-6
- LISTOPENS command on page C-6
- RELOAD command on page C-8

Changes in the H06.03 Manual

- FUP allows you to license 800 coded (TNS/E) files and to revoke 800 coded (TNS/E) licensed files. See LICENSE (Super ID) Guidelines on page 2-116.
- A correction to the MAXEXTENTS parameter ALTER command was added. See ALTER on page 2-9.
- Added the TRUST command to FUP commands.
- These file attributes were added to Appendix C, FUP Command Syntax Summary:
  - TRUSTED
  - TRUSTME
  - TRUSTSHARED
- Added a general guideline to the DUP[LICATE] command in FUP commands.
- Added command names to guidelines and examples.
- Added the TRUST variable to Example 2-6, DETAIL Format for SQL Tables and Indexes and for Enscribe and OSS Files, on page 2-98.

Changes in the G06.23 Manual

- INFO Listing Format on page 2-84 and on page 2-97 were updated for SQL/MX objects.
- `FORMAT 1|FORMAT 2` was changed to `FORMAT1|FORMAT2` in Appendix C, FUP Command Syntax Summary and Qualified File Sets on page 1-11.
- Examples were added to on page 2-97, INFO STATISTICS Listing Format on page 2-111, and INFO EXTENTS Listing Format on page 2-114.
- Handling SQL/MX Files on page 1-27 was added.
- Table 2-2, System File Code Definitions, on page 2-87 was updated.
- The SHARE option for RELOAD on page 2-146 was made more clear.

About This Manual

The File Utility Program (FUP) is a component of the standard RVU. This reference manual provides an overview of the FUP software and presents the detailed syntax for its commands.

This manual will help you manage disk files, nondisk devices (printers, terminals, and tape drives), and processes (programs) running on an HP NonStop™ server. As a reader of this manual, you should be familiar with the Guardian file-system terminology.

Organization of This Manual

Section or Appendix	Title	Description
1	FUP Overview	Provides an overview of the FUP software. The overview includes instructions for starting or interrupting FUP processes, entering FUP commands, and specifying files within FUP. The section defines FUP concepts that are common to most of the FUP commands, including file sets and list files. This section also demonstrates how to use DEFINEs with FUP and how to handle HP NonStop SQL/MP and HP NonStop SQL/MX.
2	FUP Commands	Presents the syntax for each of the FUP commands.
3	FUP Messages	Explains the error and warning messages issued by FUP and supplies the recommended recovery methods for each incident.
A	DEFINE Tables	Contains reference tables that summarize how FUP input and output options work with DEFINEs.
B	FUP Command Summary	Categorizes all the FUP commands described in Section 2.
C	FUP Command Syntax Summary	Provides a syntax summary of all FUP commands.

Additional Information

For more information about FUP and its associated components, see:

Manual	Description
5200 Optical Storage Facility (OSF) Reference Manual	Describes the FUP commands that require special considerations when used with the 5200 OSF (for D-series RVUs only)
Enscribe Programmer's Guide	Provides descriptions of structured disk files and the file attributes specified in FUP commands.
File Utility Program (FUP) Management Programming Manual	Explains how to execute commands from within an application program by using the Subsystem Programmatic Interface (SPI).
Guardian Disk and Tape Utilities Reference Manual	Explains tape handling procedures and describes the BACKCOPY, BACKUP, DCOM, DSAP, RESTORE, and TAPECOM utilities.
Guardian Procedure Errors and Messages Manual	Describes any file-system (or other) errors referenced by number in the FUP error messages.
Guardian User's Guide	Provides task-oriented instructions for using FUP and DEFINEs. It also includes a basic FUP introduction.
SQL/MP Reference Manual	Describes the SQLCI commands used on SQL files. Because this function is similar to the FUP process, this manual also describes using FUP at the SQLCI prompt.
SQL/MX Reference Manual	Describes the syntax of SQL language elements.
Safeguard User's Guide	Describes the basic security tasks that FUP requires.
TACL Reference Manual	Describes the RUN command options and all the other TACL commands and functions.
SQL/MX Glossary	Describes terms specific to SQL/MX.

Notation Conventions

Hypertext Links

Blue underline is used to indicate a hypertext link within text. By clicking a passage of text with a blue underline, you are taken to the location described. For example:

> This requirement is described under Backup DAM Volumes and Physical Disk Drives on page 3-2.

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

computer type. `Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. *`Italic computer type`* letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

```
pathname
```

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num  ]
   [ -num ]
   [ text ]
```

```
K [ X | D ] address
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }
```

```
ALLOWSU { ON | OFF }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis. An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address [ , new-value ]...

[ - ] {0|1|2|3|4|5|6|7|8|9}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char..."
```

Punctuation. Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;

LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE

   [ , attribute-spec ]...
```

Notation for Messages

This list summarizes the notation conventions for the presentation of displayed messages in this manual.

Bold Text. Bold text in an example indicates user input typed at the terminal. For example:

```
ENTER RUN CODE
?123
CODE RECEIVED:      123.00
```

The user must press the Return key after typing the input.

Nonitalic text. Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

lowercase italic letters. Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

```
p-register
```

```
process-name
```

[ ] Brackets. Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
proc-name trapped [ in SQL | in SQL file system ]
```

{ } Braces. A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
obj-type obj-name state changed to state, caused by
{ Object | Operator | Service }
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign. A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
%B101111
```

```
%H2F
```

```
P=%p-register E=%e-register
```

Notation for Management Programming Interfaces

This list summarizes the notation conventions used in the boxed descriptions of programmatic commands, event messages, and error lists in this manual.

UPPERCASE LETTERS. Uppercase letters indicate names from definition files. Type these names exactly as shown. For example:

```
ZCOM-TKN-SUBJ-SERV
```

lowercase letters. Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

```
token-type
```

!r. The !r notation following a token or field name indicates that the token or field is required. For example:

```
ZCOM-TKN-OBJNAME       token-type ZSPI-TYP-STRING.          !r
```

!o. The !o notation following a token or field name indicates that the token or field is optional. For example:

```
ZSPI-TKN-MANAGER       token-type ZSPI-TYP-FNAME32.         !o
```

Change Bar Notation

Change bars are used to indicate substantive differences between this manual and its preceding version. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:

> The message types specified in the REPORT clause are different in the COBOL environment and the Common Run-Time Environment (CRE).
>
> The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

1 FUP Overview

The File Utility Program (FUP) is a component of the standard RVU. FUP software is designed to help you manage disk files, nondisk devices (printers, terminals, and tape drives), and processes (running programs) on a NonStop system. You can use FUP to create, display, and duplicate files, load data into files, alter file characteristics, and purge files.

FUP supports these types of Enscribe disk files:

- Key sequenced
- Entry sequenced
- Relative
- Unstructured (including text files)

FUP provides information about these types of HP NonStop Open System Services (OSS), SQL/MX, and SQL/MP files:

- Tables
- Indexes
- Partitions
- Views
- Object programs

FUP supports fully qualified SQL/MX ANSI names for the commands - LISTLOCKS, LISTOPENS, INFO, and RELOAD. The syntax for the ANSI names is same as mentioned in the Unified Syntax Proposal. FUP reports information only about the named base table, and not dependent objects.

FUP has a limitation that the command line cannot exceed 132 characters. To use longer ANSI names, FUP users need to:

1. Use MXCI SHOWDDL or MXCI SHOWLABEL to get the Guardian names.
2. Run FUP on one or more of the Guardian names.

Note. FUP does not support mixing of both SQL/MX objects (using their ANSI names) and Guardian objects in the same command. Therefore, the user must use two separate commands, one for Guardian names and another for ANSI names.

FUP supports Storage Management Foundation (SMF) logical file names.

Note. For more information about Enscribe files and structured query language (SQL) files, see the *Enscribe Programmer's Guide*, the *SQL/MP Reference Manual*, the *SQL/MX Reference Manual* , and the *Guardian User's Guide .*

Starting a FUP Process

Access to FUP is available through the HP Tandem Advanced Command Language (TACL), the standard command interface in the Guardian environment. To start a FUP process, use any of these methods:

- Enter FUP, followed by a FUP command, at the TACL prompt.
- Enter FUP and then use the FUP commands interactively.
- Enter FUP, followed by a command file (followed by a TACL IN run option).

Note. For information about entering FUP commands, see Entering a FUP Command on page 1-5. For descriptions of each FUP command and its corresponding syntax, see Section 2, FUP Commands. For a description of command files and run options, see From a Command File on page 1-3.

At the TACL Prompt

To enter FUP commands at the TACL prompt, type the term FUP, followed by the command you want, and press the RETURN key:

```
1> FUP INFO *
```

The FUP command (INFO *) in this example instructs FUP to list the file information of each file in your current subvolume. After FUP executes the command, control of the terminal returns to TACL. A separate FUP process starts and completes for each command you enter.

Interactively

Entering FUP commands within FUP (interactively) saves time if you are going to enter a series of commands. Type the term FUP (without any commands or options) at the TACL prompt, and press the RETURN key to start an interactive FUP process:

```
1> FUP
File Utility Program - T6553D45 - (13OCT2000)    SYSTEM \WEST
Copyright Tandem Computers Incorporated 1981, 1983, 1985-2000
-
```

The FUP process is ready to receive commands interactively when its sign-on banner and prompt (a hyphen) are first displayed. You can then type a FUP command at each subsequent FUP prompt:

```
-INFO *
```

After FUP executes the command, the FUP prompt reappears. Type another FUP command at the subsequent prompt, or use the EXIT command to return control of the terminal from FUP to TACL:

```
-EXIT
```

From a Command File

Starting a FUP process with a command file is useful if you frequently run the same series of FUP commands. You can also process FUP commands within an interactive FUP session using the OBEY command.

Command Files

To create a command file that contains FUP commands, use a text editor (such as TEDIT). This example shows a command file (ALLSUBS):

```
-- FUP commands for obtaining a list of all
-- subvolumes in $DISK1, $DISK2, and $DISK3
--
-- Last modified 5/17/01
--
SUBVOLS $DISK1    -- Contains manufacturing files
SUBVOLS $DISK2    -- Contains administrative files
SUBVOLS $DISK3    -- Contains all other files
```

The example command file (ALLSUBS) uses a FUP command (SUBVOLS) to list the subvolumes in three different disk volumes ($DISK1, $DISK2, and $DISK3). It also includes comment lines to help identify the file and explain the operations that are performed.

Note. You can enter two dashes (--) or a less-than symbol (<) to indicate a comment. FUP ignores any text that follows these punctuation marks until the end of the command line.

Run Options

The TACL environment includes a set of predefined commands—including the TACL RUN command. You must use a run option when you start a FUP process with a command file. The two run options that FUP uses most often are IN (for specifying an input file) and OUT (for an output file).

Note. For more information about the TACL RUN command, see the *TACL Reference Manual*.

Type the term FUP and the command file and run option you want, and press RETURN:

```
1> FUP / IN ALLSUBS /
```

In this example, the FUP process starts when you use a command file (ALLSUBS) and run option (IN). The command in this example writes to the terminal because there is no OUT file. After FUP executes the last command in the command file, control of the terminal returns to TACL.

△ **Caution.** For software product revisions (SPRs) earlier than T6553ABQ, FUP can cause a processor halt if it receives a bad startup message. This situation occurs mainly when FUP is started programmatically and the FUP STARTUPMESSAGE is accidentally corrupted, or when FUP is started through SCF after the “DEFAULT STARTUP MESSAGE” gets corrupted when a `SYSTEM \system-name` is executed. To avoid this situation, install T6553ABQ or a later SPR.

Using FUP Custom Files

FUP reads two files (FUPLOCL and FUPCSTM) before it issues its first prompt. This allows you to create a customized FUP environment before entering any commands. Both these files are standard FUP command files that contain ASCII text with valid FUP commands.

FUP Custom File Guidelines

- The FUPLOCL file must be in the current system subvolume, $SYSTEM.SYS*nn* (or $SYSTEM.SYSTEM). FUPLOCL lets you set a site-standard FUP environment.
- The FUPCSTM file must be in the user’s logon subvolume. FUPCSTM lets individual users set their own FUP environment.
- Although FUP executes each command in the FUPLOCL and FUPCSTM files and can execute any FUP command, custom files are most useful for setting environment commands (such as CONFIGURE and SET) for a FUP session.
- FUP executes any commands from the FUPLOCL file (if present) before it executes any commands from the FUPCSTM file (if present).
- If you are running FUP on a remote node, FUP looks for the custom files in the appropriate subvolumes on the remote node.

- By default, FUP does not echo the commands for either file. To start echoing, specify CONFIGURE ECHO OBEY in either file.
- If there are no existing custom files when FUP begins, default custom files are automatically created for your security.
- Custom files must be EDIT files.
- Errors encountered during the processing of a custom file could cause the FUP session to terminate.

FUP Custom File Example

To save the current CONFIGURE options to a file that can be executed with FUP OBEY or as a FUPCSTM file:

```
-SHOW /OUT FUPCSTM/ CONFIGURE AS COMMANDS
```

Interrupting or Terminating a FUP Process

To terminate a FUP process after it has started, use CTRL-Y.

To interrupt a FUP process, press the BREAK key.

User Entry	Results
CTRL-Y	Terminates a FUP process from the FUP prompt or a PURGE command from the PURGE prompt, and stops the execution of COPY and LOAD commands during input from the terminal.
BREAK	Aborts any FUP commands that generate listings (including the FILES, INFO, SUBVOLS, and COPY commands), and the PURGE command prompts. Three responses are possible: ● FUP prompt—FUP recognized the break and terminated the command. ● TACL prompt—TACL recognized the break, but FUP continues in the background. ● Nothing or a delay—FUP cannot break when BREAK is pressed. FUP should recognize the BREAK after a brief period. If FUP does not stop, use the TACL STOP command from another TACL session. Pressing the BREAK key while the FUP prompt is displayed or while FUP is executing a nonlisting command, returns control of the terminal to TACL—but the FUP process continues.

Entering a FUP Command

For descriptions of the syntax for each FUP command, see Section 2, FUP Commands. The basic command structure (including run options) is:

```
FUP [ / run-options / ] [ command ]
```

run-options

are any of the available options for the TACL RUN command. They must be separated from each other by commas, and enclosed on the command line with slashes (/). Although each option is available when you run FUP from the TACL prompt, only one (the OUT run option) is also available within FUP. These run options are used most often with the FUP process:

IN *filename*

names a disk file, nondisk device, or process from which FUP reads commands. Any IN disk file that you specify must be an EDIT file, unstructured file (132 byte records), or Enscribe-structured file. You cannot specify an SQL file as an IN disk file.

If you omit this option, FUP uses the IN file name that is in effect for the current TACL process— usually the home terminal.

OUT *listfile*

names a nondisk device, process, or a disk file to which FUP directs its listing output (unless you use the OUT option of a subsequent FUP command to direct the output somewhere else). You must use a Guardian file name, a SPOOL DEFINE, or MAP DEFINE as the OUT list file to run FUP. Any specified OUT disk file must be an EDIT file, unstructured file, or Enscribe-structured file. You cannot specify an SQL file as an OUT disk file. You can use this option to specify output to a TACL variable.

If you omit this option, FUP uses the OUT list file that is in effect for the current TACL process—which is usually the home terminal.

If *listfile* does not exist, FUP creates it as an EDIT file with a maximum record length of 132 characters.

If *listfile* does exist, FUP appends output to it.

If *listfile* is an unstructured disk file, each of its records is 132 characters long, and any partial lines are blank-filled to column 132.

For a description of using a SPOOL DEFINE for *listfile*, see Using DEFINEs With FUP on page 1-21.

command

is a FUP command. You can enter only one FUP command in a TACL command. If you need additional screen space to enter the command, end the command line with an ampersand (&) and press RETURN. Continue the command on the next line at the new TACL prompt.

If you type FUP followed by *command*, the FUP process terminates after executing the command and returns control of the terminal to TACL.

If you type FUP without *command*, you must terminate the FUP process using the EXIT command after you finish your FUP activity.

FUP Command Guideline

If you enter conflicting options, FUP scans the options and uses the last one entered:

```
DUP A, B, OLD, NEW, SOURCEDATE, SAVEID
```

The FUP process uses the NEW option (not the OLD option) and the SAVEID option (not the SOURCEDATE option). If you use related options (such as SAVEALL, SAVEID, or SOURCEDATE), FUP uses the last option entered and cancels the previous entry.

FUP Command Examples

- To use a command file (ALLSUBS) with the IN run option and direct the results of this activity (output) to the LIST1 disk file using the OUT run option:

  ```
  TACL1> FUP / IN ALLSUBS, OUT LIST1 /
  TACL2>
  ```

  FUP reads the input from ALLSUBS, processes all of its commands, sends the output to LIST1, and then returns to a new TACL prompt.

- To start an interactive FUP process using the DUP command (with the PURGE option) to purge the contents of FILE2 before duplicating the contents of FILE1 into FILE2 (the two files in this example, FILE1 and FILE2, are from the default volume and subvolume):

  ```
  TACL1> FUP
  -DUP FILE1, FILE2, PURGE

     (FUP displays information about the process)
          ...
  ```

  While the command is executing, FUP displays any information about the ongoing process and then returns you to the FUP prompt (hyphen) when the process is complete.

- To avoid the interactive FUP process shown in the previous example for the DUP command (and the PURGE option) at the TACL prompt, and purge FILE2 before it duplicates the contents of FILE1 to FILE2:

  ```
  TACL1> FUP DUP FILE1, FILE2, PURGE

     (FUP displays information about the DUP operation)
          ...
  TACL2>
  ```

  After FUP displays information about the DUP operation, it returns to a new TACL prompt when the process is complete.

Specifying Files

FUP commands make it easy to:

- Create, display, and duplicate files
- Load data into files
- Alter file characteristics
- Purge files

Before you use FUP to create or manage files, become familiar with the various file types and the methods used to specify them.

Note. For more information on the different types of Enscribe-structured files, see the *Enscribe Programmer's Guide*.

The different types of files in the Guardian environment include disk files (containing data, code, or text), nondisk devices (terminals, printers, or tape drives), spooler files (code 129), and processes (programs that are running). Disk file names have four parts: node, volume, subvolume, and file identifier. The names of nondisk devices and processes must begin with a dollar sign ($), followed by one to five alphanumeric characters (and additional qualifiers if applicable).

Note. For more information on file-naming conventions, see the *Guardian User's Guide* .

Three common syntax terms appear throughout the FUP commands. Each of these terms provides a way to specify the file or files you want to affect with a FUP command. The syntax terms are:

- *`listfile`*
- *`fileset-list`*
- *`fileset`*

Note. For information on when FUP commands do not apply to SQL files, see the command descriptions in Section 2, FUP Commands.

Listfile Parameter

The *`listfile`* parameter refers to a nondisk device, a process, an existing disk file, or a spooler file (code 129) to which you direct the output of a FUP command. It always appears with the OUT keyword in the syntax descriptions for the TACL RUN command and in the FILENAMES, FILES, HELP, INFO, LISTLOCKS, LISTOPENS, RELOAD, SHOW, STATUS, SUBVOLS, SUSPEND, and VOLS commands for FUP.

If you omit this option, FUP uses the OUT list file that is in effect for the current TACL process—usually the home terminal.

If a list file does not exist, FUP creates it as an EDIT file with a maximum record length of 132 characters.

If a list file does exist, FUP appends output to it.

If the list file is an unstructured disk file, each of its records is 132 characters long, and any partial lines are blank-filled to column 132.

You can specify a SPOOL DEFINE for *listfile*. For a description of using a SPOOL DEFINE, see Using DEFINEs With FUP on page 1-21.

If you enter only a partial file name, FUP expands it using the current default node, volume, and subvolume names.

Fileset-list Parameter

The *fileset-list* parameter refers to one or more file sets. It can contain up to ten different sets of files enclosed in parentheses. The form for *fileset-list* is:

```
{ fileset | ( fileset [ , fileset ] ... ) }

fileset is:
[[[ \node.] $volume. ] subvolume. ] file-id [ qualified-expr
]
```

Fileset-list Examples

- To specify the files MYFILE, MYSRC, and MYOBJ in the current default subvolume on the current default volume:

  ```
  (MYFILE, MYSRC, MYOBJ)
  ```

- To specify all files in the current default subvolume and all files in all the subvolumes on the volume $VOL1:

  ```
  (*, $VOL1.*.*)
  ```

- To specify all files in all subvolumes on the current default volume and all files on the volume $VOL2:

  ```
  (*.*, $VOL2.*.*)
  ```

Fileset Parameter

The *fileset* parameter refers to a set of files. It can be one file, all the files in a subvolume, all the files on a volume, a subset specified by the wild-card option, or a subset produced by a qualified file set. The file or files can be fully qualified or can include wild cards or qualifiers. The form for *fileset* is:

```
[[[ \node.] $volume. ] subvolume. ] file-id [ qualified-expr
]
```

node

is the name of a node in an Expand network. If you omit *node*, FUP uses the current default node: either the node that was in effect when you started FUP or the node you specified during the last FUP SYSTEM or VOLUME command.

volume

is the name of a volume (disk drive). If you omit *volume*, FUP uses the current default volume: either the volume that was in effect when you started FUP or the volume you specified during the last FUP VOLUME (or SYSTEM) command. If you specify *volume*, you must also specify *subvolume*.

subvolume

is the name of the subvolume. It consists of alphanumeric characters, wild-card characters (* or ?*), or a combination of both. If you omit *subvolume*, FUP uses the current default subvolume: either the subvolume that was in effect when you started FUP or the subvolume you specified during the last FUP VOLUME (or SYSTEM) command.

For compatibility with previous versions of FUP, the subvolume name can be the default in most commands. If the subvolume is omitted, FUP uses the current default subvolume and issues a warning that this option might be deleted in future versions of FUP.

file-id

is the name of an Enscribe or SQL disk file. It consists of alphanumeric characters, wild-card characters (* or ?), or a combination of both. Table 1-2 on page 1-25 shows the interaction between FUP commands and SQL files.

qualified-expr

specifies additional restrictions to a file set. It extends the power of the wild-card option in file names by specifying attributes of the file. A file set with *qualified-expr* is called a qualified file set. For more information, see Qualified File Sets on page 1-11.

Wild-Card Option

You can use asterisks (*) or question marks (?) as wild-card characters to help specify files. The asterisk can represent from zero through eight unspecified characters in the position you place it. The question mark represents only one character in the position you place it.

Wild-Card Guidelines

- The only FUP commands that allow you to use wild-card characters to specify a volume name are VOLS, SUBVOLS, INFO, FILENAMES, and FILES.

- You cannot use more than eight characters (including wild-card characters) in any portion of the file name (volume, subvolume, or file identifier).
- The only valid use of the wild-card option in a destination file-set specification is a single asterisk (*) in the subvolume or file ID position. The DUP MYFILE, * command is valid, but the DUP MYFILE, MY* command is not.

Wild-Card Examples

- To specify the file MYFILE on the current FUP default volume and subvolume:

  ```
  MYFILE
  ```

- To specify all the file IDs that begin with MY on the current default volume and subvolume:

  ```
  MY*
  ```

- To specify all the file IDs that begin with MY, are followed by one character, and end with ILE on the current default volume and subvolume:

  ```
  MY?ILE
  ```

- To specify all the files in the subvolume MYSVOL on the current default volume:

  ```
  MYSVOL.*
  ```

- To specify all the files in all the subvolumes on the volume $VOL1:

  ```
  $VOL1.*.*
  ```

- To specify all the files in all the subvolumes on the current default volume:

  ```
  *.*
  ```

- To specify all the files in the subvolume MYTOWN on the volume $BRDWAY on the \NY node:

  ```
  \NY.$BRDWAY.MYTOWN.*
  ```

Qualified File Sets

A file set with *qualified-expr* is called a qualified file set. A qualified file set specifies an additional set of restrictions to a file set. It defines a subset (to *fileset*) and extends the power of the wild-card option in file names by including arbitrary qualifications for file attributes.

Note. File sets can use qualified file-set syntax except where explicitly prohibited in the descriptions of each command in Section 2, FUP Commands.

The form for *qualified-expr* is:

```
qualifier [ ,qualifier ] ...


qualifier is:

    EXCLUDE fileset
    FROM CATALOG[S] catalog-list
    START file-id
    WHERE expression


catalog-list is:

    [ \node.]$volume [.subvol ] | subvol


expression is:

    ( expression )
    expression AND expression
    expression OR expression
    NOT expression
    file-attribute
    OWNER = user-id
    timestamp-field time-conditional time-value
    FILECODE conditional-number
    EOF conditional-number
```

file-attribute is:

```
    ALTKEY
    AUDITED
    BROKEN
    COLLATION
    CORRUPT
    CRASHOPEN
    ENSCRIBE
    ENTRYSEQUENCED
    FORMAT1|FORMAT2
    INDEX
    [ SHORTHAND | PROTECTION ]  VIEW
    KEYSEQUENCED
    LICENSED
    OPEN
    PROGID
    RELATIVE
    [ PRIMARY | SECONDARY ]  PARTITION
    ROLLFORWARDNEEDED
    SAFEGUARD
    SQL
    SQLPROGRAM
    TABLE
    UNSTRUCTURED
```

user-id is:

```
    group-name.user-name
    group-name.*
    group-number, user-number
    group-number,*
```

timestamp-field is:

```
    CREATIONTIME
    EXPIRATIONTIME
    LASTOPENTIME
    MODTIME
```

time-conditional is:

```
    AFTER  | >
    BEFORE | <
```

```
time-value is:

    [ date ] time
    [ time ] date

time is:

    hh:mm[:ss]


date is:

    dd mmm yyyy
    mmm dd yyyy
```

qualifier

is qualifying criteria that you can specify to include files or objects in a file set. You can specify qualifiers in any order, but use each qualifier only once per file-set list.

EXCLUDE *fileset*

excludes the files you specify from a process. This specification is identical to the NOT option. You can include wild-card characters in this specification.

FROM CATALOG[S] *catalog-list*

includes SQL objects from *catalog-list* that match the file-set list.

catalog-list

specifies a volume and subvolume (or a DEFINE name of a file) containing descriptions of the SQL objects (tables, indexes, views, or SQL programs).

START *file-id*

specifies a starting position within a file set or file-set list. The process starts on the first file after the file ID you specify. Although the wild-card option is not valid for this specification, the file ID does not have to correspond to an existing file. This option is useful for restarting a process that was interrupted.

WHERE *expression*

provides additional qualifying criteria for files or objects. The order of precedence in evaluating expressions is parentheses ( ), NOT, AND, and OR.

file-attribute

is an additional parameter to the WHERE qualifier. You must use the complete parameter name in the command. Abbreviations are not permitted.

ALTKEY

adds, replaces, or alters an alternate-key specification.

AUDITED

specifies files that are audited by the HP NonStop Transaction Management Facility (TMF).

BROKEN

specifies files that are broken. The file needs media recovery because an I/O or consistency check failure occurred the last time it was open. A good example of its use is: `WHERE NOT BROKEN`.

COLLATION

specifies an SQL collation object.

CORRUPT

specifies any files with questionable contents. The destination file is marked CORRUPT automatically by FUP during DUP or LOAD operations. If the DUP or LOAD operation is completed abnormally, the CORRUPT flag remains set for the file. A good example of its use is: `WHERE NOT CORRUPT`.

CRASHOPEN

specifies files not closed normally by the disk process. A good example of its use is: `WHERE NOT CRASHOPEN`.

ENSCRIBE

specifies Enscribe files.

ENTRYSEQUENCED

specifies entry-sequenced files.

FORMAT1

describes files that are smaller than 2 GB minus 1 MB size limit. A Format 1 file cannot exceed the 2 GB minus 1 MB size limit.

FORMAT2

describes files that are bigger than 2 GB minus 1 MB size limit. A Format 2 file can exceed the 2 GB minus 1 MB size limit.

Note. The Format 1 files were created on systems running RVUs preceding D46.00 or G06.00. The Format 2 files were created on systems running D46.00, G06.00, or later RVUs.

`INDEX`

specifies SQL indexes.

`KEYSEQUENCED`

specifies key-sequenced files.

`LICENSED`

specifies LICENSED files. These files are executable object files that run in privileged mode.

`OPEN`

specifies files that are in the OPEN state.

`[ PRIMARY | SECONDARY ] PARTITION`

specifies the primary partition of partitioned files, the secondary partition of partitioned files, or all partitions of partitioned files (if neither primary nor secondary is specified).

`PROGID`

specifies files that have the PROGID flag set.

`RELATIVE`

specifies relative files.

`ROLLFORWARDNEEDED`

specifies any files that need an HP Transaction Management Facility (TMF) rollforward process. A good example is: `WHERE NOT ROLLFORWARDNEEDED`.

`SAFEGUARD`

specifies files protected by Safeguard at the file level.

`SQL`

specifies all types of SQL files except SQL program files.

`SQLPROGRAM`

specifies SQL program files.

`TABLE`

specifies SQL tables.

UNSTRUCTURED

specifies unstructured disk files.

[SHORTHAND | PROTECTION] VIEW

specifies the shorthand SQL views, the protection SQL views, or all the SQL views (if neither shorthand nor protection are specified).

OWNER = *user-id*

specifies user identification for the form specified.

timestamp-field

helps select a file based on a specific time:

CREATIONTIME

selects a file based on when it was created.

EXPIRATIONTIME

selects a file based on when it expires and can be purged. You must use this option to select dates set with the NOPURGEUNTIL timestamp in FUP ALTER.

LASTOPENTIME

selects a file based on when it was last opened.

MODTIME

selects a file based on when it was last modified.

time-conditional

selects a file based on a specific time:

[AFTER | >]

selects a file after the time you specify.

[BEFORE | <]

selects a file before the time you specify.

time-value

selects a file based on a specific time. The default is 00:00:00 (midnight) of the current date:

[*date* | *time*] *time* | *date*

time is:

```
hh:mm[:ss]
```

hh is the hour. *mm* is the minutes. *ss* is the seconds.

date is:

```
dd mmm yyyy
mmm dd yyyy
```

mmm is always the first three characters in the name of the month. *yyyy* must be four digits for the year. Valid examples are:

```
1 JAN 2001 06:30
JAN 1 2001 06:30
6:30 1 JAN 2001
20 OCT 1999 07:29:30
06:15 JAN 1 1998
23:30:00 DEC 31 2000
```

FILECODE *conditional-number*

selects a file based on its file code. *conditional-number* represents a file code (such as 101) and is preceded by one of these symbols:

< (less than)

<= (less than or equal to)

= (equal to)

> (greater than)

>= (greater than or equal to)

<> (not equal to)

EOF *conditional-number*

selects a file based on the number of bytes it contains. *conditional-number* represents the number of bytes (you provide) and is preceded by one of these symbols:

< (less than)

<= (less than or equal to)

= (equal to)

> (greater than)

>= (greater than or equal to)

<> (not equal to).

Note. Keywords for qualified file sets must be specified completely. Abbreviations are not permitted. For example, use KEYSEQUENCED (not K), ENTRYSEQUENCED (not E), RELATIVE (not R), UNSTRUCTURED (not U), and FILECODE (not CODE).

Note. Qualifiers appear immediately after the file set they are qualifying. This does not have to be the end of the FUP command.

Qualified File Set Examples

- To obtain information about all EDIT files that begin with the letter *S*:

```
INFO S* WHERE FILECODE=101
```

- To obtain information about all EDIT files that were created on or after 09/02/01:

```
INFO LE*.* WHERE FILECODE=101 AND MODTIME AFTER 2 SEP 2001
```

- To obtain information about all files beginning with the letters *F* through *Z* except the files that begin with the letter *S*:

```
INFO * START F EXCLUDE S*
```

- To duplicate all object files within a subvolume:

```
DUP * WHERE FILECODE=101, ANYWHERE.*
```

- To obtain information about all files in the current subvolume beginning with the letters *M* through *Z*:

```
INFO * START M
```

- To obtain information about all files in the current subvolume that are not owned by user 1,86:

```
INFO * WHERE NOT OWNER 1,86
```

Creating Files

You can create files using FUP commands.

Examples of Creating Files

- To use the FUP ALTER command to create a partitioned alternate-key file:

```
FUP
-ALTER filename, PART (sec-partition-num,
 [\node.]$volume
 [ , pri-extent-size [ , sec-extent-size] ]
```

- To create a partitioned alternate-key sequenced file using the SET command:

```
FUP
-SET TYPE K
-SET CODE 1001
-SET EXT (32,8)
-SET REC 54
-SET BLOCK 4096
-SET KEYLEN 2
-SET ALTKEY ("LO", KEYOFF 42, KEYLEN 4)
-SET ALTKEY ("VN", KEYOFF 46, KEYLEN 8)
-SET ALTFILE (0,invalt)
-SET PART (1, $ade001,5,5)
-CREATE INV
CREATED- $STORE1.SVOL1.INV
CREATED- $STORE1.SVOL1.INVALT
```

Partitioning Files

You can partition existing files using FUP commands.

Examples of Partitioning Files

- To use the LISTOPENS, SET, PURGE, CREATE, and LOADALTFILE commands to partition an existing alternate-key sequenced file:

  1. Ensure the file is closed, using the LISTOPENS command:

     ```
     -LISTOPENS file0214
     ```

  2. Set the file-creation attributes to match the existing file using the SET command:

     ```
     -SET LIKE $USA.texas.austin
     ```

  3. Set secondary partition specifications for the partitioned file using the SET command:

     ```
     -SET PART (1,$ASIA,primary EXT,secondary EXT,
     "PRaltkeyvalue1"
     -SET PART (2,$VOLnn,primary EXT,secondary EXT,
     "PRaltkeyvalue2"
     ```

  4. Purge the existing alternate-key file:

     ```
     -PURGE $VOLnn.subvol.filename
     ```

  5. Create the new partitioned alternate-key file:

     ```
     -CREATE $VOLnn.subvol.partfilename
     ```

  6. Load the new partitioned alternate-key sequence file:

     ```
     -LOADALTFILE 0, $VOL.subvol.partfilename
     ```

  Repeat these commands for each file you want to partition.

Related Commands

COMMAND	Function
CREATE	Creates a file using the current file-creation parameter values that have been defined with a SET command
PURGE	Deletes a file
LOADALTFILE	Creates an alternate-key file from a primary file
LISTOPENS	Lists all processes that now have one or more designated files open

Using DEFINEs With FUP

You can use SPOOL, MAP, or TAPE DEFINEs to specify information for a FUP process before you start it:

DEFINE Type	Usage
SPOOL DEFINE	Sends command output to a spooler
MAP DEFINE	Substitutes a logical name for an actual file name
TAPE DEFINE	Sends command output to a tape file or receives a tape file as input

DEFINE names on NonStop systems always begin with the “=” character.

Note. For more information about DEFINEs, see the *Guardian User’s Guide .*

SPOOL DEFINEs

You can specify a SPOOL DEFINE in these situations:

- As *`listfile`* with the OUT option in:
  - A TACL RUN command
  - The FILENAMES, FILES, HELP, INFO, LISTLOCKS, LISTOPENS, RELOAD, SHOW, STATUS, SUBVOLS, SUSPEND, and VOLS commands
- As the destination file in a COPY or BUILDKEYRECORDS command

When you use a SPOOL DEFINE with a FUP command, output from the command is spooled to the spooler location specified with the LOC attribute in the DEFINE. If this location is a printer, FUP command output is queued and then printed. Otherwise, the output remains in the spooler, where you can use PERUSE to view, redirect, or delete it.

SPOOL DEFINEs Example

```
COPY ricecake,=MY_PRINTER
```

MAP DEFINEs

You can specify a MAP DEFINE wherever FUP permits a file name. It is sometimes easier to use a DEFINE name such as =CUSTOMERS than an actual file name such as \SF.$ACCNTS.CURRNT.CUSTNMES.

MAP DEFINEs Example

```
DUP cereal,=MY_DEFINE
INFO =MY_DEFINE
```

TAPE DEFINEs

You can specify a TAPE DEFINE as:

- *out-filename* (destination) in a BUILDKEYRECORDS command
- *in-filename* (source) or *out-filename* (destination) in a COPY command
- *in-filename* (source) in a LOAD command

Note. Do not use TAPE DEFINE attributes that conflict with your FUP command parameters. For more information about how FUP input and output options work with TAPE DEFINEs, see Appendix A, DEFINE Tables.

Handling Different Types of Files

This section contains information about handling file formats, as well as guidelines for handling OSS, SQL/MP, SQL/MX, and SMF files.

Handling File Formats

A new disk file format (Format 2) for describing large format files (big files) is available starting with the G06.00 and D46.00 RVUs:

Format 1	A file created on the G06.00 or D46.00 RVU or later that is smaller than 2 GB minus 1 MB, or a file created on RVUs preceding G06.00 or D46.00.
Format 2	Either a large format file or a file that can contain larger partitions than a file created on RVUs preceding G06.00 or D46.00. A Format 2 file can exceed the 2 GB minus 1 MB size limit of a Format 1 file. However, you can explicitly create a Format 2 file even if its maximum partition size is less than 2 GB.

At the time of creation, you can specify the file format, as Table 1-1 shows.

Table 1-1. File Format Codes

Format Code	Maximum Block Size (in Bytes)	Maximum Record Size (in Bytes)	Maximum Partition Size (in Bytes)
0	4096	Determined by system	Determined by system
1	4096	Block size - 34 (key sequenced)	2 GB - 1 KB
		Block size - 24 (relative and entry sequenced)	
2	4096	Block size - 56 (key sequenced)	1024 GB
		Block size - 48 (relative)	
		Block size - 44 (entry sequenced)	

Note. FUP supports Format 2 SQL partitions in the G06.13 and later RVUs. FUP supports fallback from G06.13 to as far back as G06.03. If you are running an RVU prior to G06.03, you cannot fall back to your previous RVU.

FUP commands display error 584 when they encounter Format 2 SQL partitions on a system that has fallen back from G06.13 to an earlier RVU. FUP continues to process the remaining files.

When you specify Format 0, the system decides the format of the file based on other file attributes. The system chooses Format 2 when the block size is over 4 KB or when a partition size is over 2 GB minus 1 KB. For unstructured, relative, and entry-sequenced files, the system chooses Format 2 when the total file size is 4 GB or more. By default, Format 1 is assumed.

The FUP INFO command provides the value of the file format for existing files.

Moving Format 1 File Contents to Format 2

You cannot convert an existing Format 1 file partition to Format 2. Any attempt to alter the max extents of the Format 1 file, such that the partition exceeds 2 GB minus 1 MB, is rejected.

To put the content of a Format 1 file into a Format 2 file:

1. Create a Format 2 file.
2. Use the LOAD command or a comparable operation to move the contents of the Format 1 file into the Format 2 file.

   You cannot update the Format 1 file during such a LOAD operation, and all access must be shut down to switch to the new file.

You can create an equivalent Format 2 file for most Format 1 files. A Format 1 file with a record size within 56 bytes of the block size (48 if not key-sequenced) cannot have a Format 2 file with the same size created for it. You can use a larger block size for the Format 2 file unless the Format 1 file already has a 4096 byte blocksize.

You likely need to change existing applications to access Format 2 files over 4 GB in size that are not key-sequenced. These files require the use of a 64-bit primary key, and system interfaces preceding G06.00 or D46.00 support only 32-bit primary keys. Applications are unlikely to require any changes to access key-sequenced Format 2 files.

Except for key-sequenced files, all partitions of a file must have the same format (Format 1 or Format 2). You cannot create a key-sequenced file with a mix of partition formats. However, you can replace the partitions one at a time with equivalent Format 2 partitions if the block and record sizes are the same for all partitions.

You should convert a key-sequenced partition to Format 2 when it approaches its capacity limit:

1. Create a new Format 2 replacement partition on a different volume.
2. Load the data into the new partition from the Format 1 partition.
3. Change the file partition information to point to the new Format 2 partition instead of the Format 1 partition.

If Format 2 files are created on a system running D46.00, G06.00, or subsequent RVUs, fallback to an RVU prior to D46.00 or G06.00 precludes access to the contents of those files. Access to Format 2 files in an Expand network from RVUs before D46.00 or G06.00 will not be possible.

Handling OSS Files

FUP can only handle OSS files with the FUP INFO command. OSS files cannot be specified in any other FUP commands.

Handling SQL/MP Files

One of the relational database management systems that is used to define and manipulate industry-standard Structured Query Language (SQL) is SQL/MP. SQL/MP files include object programs, tables, views, indexes, partitions, and catalogs. All the FUP commands (unless noted) are applicable to the SQL/MP object program files.

Note. The FUP SET, RESET, and SHOW commands do not apply to SQL/MP files. They apply only to Enscribe files. The FUP control commands described in Appendix B, FUP Command Summary, do not operate on SQL/MP files.

Table 1-2 shows the FUP commands that do not apply to all the different types of SQL/MP files and lists any alternative methods for manipulating the files.

Table 1-2. FUP Commands and SQL/MP Files (page 1 of 2)

	Applicable to SQL/MP File Type					
FUP Command	**Table**	**View**	**Index**	**Partition**	**Catalog**	**SQLCI Equivalent**
ALLOCATE	No	No	No	No	No	CREATE TABLE CREATE INDEX ALTER TABLE ALTER INDEX
ALTER	No	No	No	No	No	ALTER TABLE ALTER INDEX
BUILDKEYRECORDS	No	No	No	No	No	CREATE TABLE
CHECKSUM	Yes	No	Yes	Yes	No	
COPY	No	No	No	No	No	COPY utility
CREATE	No	No	No	No	No	CREATE TABLE CREATE INDEX
DEALLOCATE	No	No	No	No	No	ALTER TABLE ALTER INDEX
DUP	No	No	No	No	No	DUP utility
FILES	Yes	Yes	Yes	Yes	Yes	
FILENAMES	Yes	Yes	Yes	Yes	Yes	
GIVE	No	No	No	No	No	ALTER command SECURE utility
INFO	Yes	Yes	Yes	Yes	Yes	FILEINFO utility
LICENSE	No	No	No	No	No	
LISTOPENS	Yes	1	Yes	Yes	Yes	
LOAD	No	No	No	No	No	LOAD utility
LOADALTFILE	No	No	No	No	No	LOAD utility
PURGE	No	No	No	No	No	DROP command PURGE utility
PURGEDATA	No	No	No	No	No	
RELOAD	Yes	No	Yes	Yes	Yes	

1 = Yes for protection views; No for shorthand views

Table 1-2. FUP Commands and SQL/MP Files (page 2 of 2)

	Applicable to SQL/MP File Type					
FUP Command	**Table**	**View**	**Index**	**Partition**	**Catalog**	**SQLCI Equivalent**
RENAME	No	No	No	No	No	
REVOKE	No	No	No	No	No	
SECURE	No	No	No	No	No	ALTER command SECURE utility
SUBVOLS	Yes	Yes	Yes	Yes	Yes	

1 = Yes for protection views; No for shorthand views

SQL/MP File Guidelines

- If you use a FUP command that supports only SQL/MP object program files and it encounters an SQL/MP table or view, the SQL/MP table or view is ignored, and a warning message appears.
- Although you can use FUP commands to provide information about SQL/MP files, the commands cannot manipulate or duplicate the SQL/MP files.
- To display information for the SQL/MP files and SQL/MP object program files, use the FUP FILES, FILENAMES, INFO, LISTOPENS, and SUBVOLS commands.
- Although most FUP commands apply only to the SQL/MP object program files, you can execute any of the commands directly from the SQL/MP Conversational Interface (SQLCI).
- If you use the FUP DUP command to duplicate an SQL/MP object program file, the SQL/MP SENSITIVE and SQL/MP VALID flags in the file label of the duplicate copy are turned off, and a warning message indicates that you must compile the file again with the SQL/MP compiler.
- The only way to SQL/MP license an SQL/MP object program file is to apply the FUP LICENSE command to the file.
- Although you can use the FUP CHECKSUM command to handle SQL/MP tables and indexes, it does not apply to SQL/MP views (protection and shorthand). SQL/MP views are skipped, and a warning message appears during a FUP CHECKSUM process.
- To display information about Enscribe and SQL/MP files, use the FUP INFO command or the SQLCI FILEINFO utility. Although these two methods have the same function and listing formats, the FUP INFO *fileset-list* parameter does not support the FROM CATALOG option or any DEFINE specifications.
- The SQL/MP LOAD utility is equivalent to both FUP LOAD and FUP LOADALTFILE. When you use SQL/MP LOAD to load an SQL/MP table, all

indexes that depend on the table are loaded automatically. No SQL/MP commands load indexes directly.

- For compatibility with the FUP PURGE command, substitute the keyword PURGE for DROP in the SQL/MP DROP utility.

Handling SQL/MX Files

SQL/MX is a relational database management system that promotes efficient online access to large distributed databases.

Table 1-3. FUP Commands and SQL/MX Files

	Applicable to SQL/MX File Type				
FUP Command	**Table**	**Index**	**Resource Fork**	**Metadata Tables**	**MXCI Equivalent**
ALLOCATE	No	No	No	No	
ALTER	No	No	No	No	
BUILDKEYRECORDS	No	No	No	No	
CHECKSUM	Yes	Yes	Yes	Yes	
COPY	No	No	No	No	
CREATE	No	No	No	No	CREATE TABLE
DEALLOCATE	No	No	No	No	
DUP	No	No	No	No	
FILES	Yes	Yes	Yes	Yes	
FILENAMES	Yes	Yes	Yes	Yes	
GIVE	No	No	No	No	
INFO	Yes	Yes	Yes	Yes	MXCI SHOWLABEL
LICENSE	No	No	No	No	
LISTOPENS	Yes	Yes	Yes	Yes	
LOAD	No	No	No	No	
LOADALTFILE	No	No	No	No	
PURGE	No	No	No	No	MXCI DROP utility
PURGEDATA	No	No	No	No	
RELOAD	Yes	Yes	Yes	Yes	
RENAME	No	No	No	No	
REVOKE	No	No	No	No	
SECURE	No	No	No	No	
SUBVOLS	Yes	Yes	Yes	Yes	

SQL/MX File Guidelines

- Although you can use FUP commands to provide information about SQL/MX files, the commands cannot manipulate or duplicate the SQL/MX files.
- To display information for SQL/MX files, use the FUP FILES, FILENAMES, INFO, LISTOPENS, and SUBVOLS commands.
- The partition information for SQL/MX objects is displayed without the partition key. The partition key information for SQL/MX objects is available from MXCI SHOWDDL.
- For compatibility with the FUP PURGE command, substitute the keyword PURGE for DROP in the MXCI DROP utility.

Handling SMF Files

Storage Management Facility (SMF) provides location-independent naming capabilities for Guardian files. FUP supports SMF commands by displaying information about files on virtual disks and lets you specify physical volumes.

2 FUP Commands

This section describes each FUP command, including:

- A summary of the command function
- The command syntax, including its parameters and variables
- The format of the command output (if applicable)
- Guidelines for using the command
- Examples of using the command
- A list of related commands

Before you use FUP commands, you should be familiar with the various Guardian file types and the different methods to specify them. For instructions on using the different file types, see Specifying Files on page 1-8.

Each FUP command belongs to one of four command groups (control, information, security, and management) that are categorized according to their function. For quick reference of these command groups, see Appendix B, FUP Command Summary.

FUP Command	Description	Page (page 2 of 3)
DISPLAYBITS	Lets COPY, INFO, and SHOW display 8-bit characters	2-61
DUP[LICATE]	Copies Enscribe disk files	2-61
EXIT	Stops FUP and returns you to the command interpreter	2-70
FC	Lets you modify and reexecute an existing command	2-70
FILENAMES	Lists files that match specified wild-card options	2-72
FILES	Lists all files associated with specified subvolumes	2-74
GIVE	Changes the owner of a file	2-76
HELP	Lists the syntax of all FUP commands	2-78
HISTORY	Displays your previous FUP commands	2-79
INFO	Displays disk file characteristics of various files or tables	2-80
LICENSE (Super ID)	Lets nonprivileged users execute privileged programs	2-115
LISTLOCKS	Displays information on all locks for a specified file set	2-116
LISTOPENS	Lists processes that have any of the specified files open	2-122
LOAD	Loads data into a structured disk file without affecting associated alternate-key files	2-129
LOADALTFILE	Generates alternate-key records for a designated alternate-key file and then loads the records into the file	2-136
OBEY	Reads and executes commands from the specified file	2-138
PURGE	Deletes selected files or file sets	2-139
PURGEDATA	Removes all data from a file	2-144
RELOAD	Reorganizes a key-sequenced file or an SQL table or index	2-146
RELOCATE	Moves files on virtual disks between physical volumes	2-151
RENAME	Changes the file or subvolume name of a disk file	2-152
REPORTWIDTH	Sets the maximum length for FUP to format its output	2-154

FUP Command	Description	Page (page 3 of 3)
RESET	Restores file-creation attributes to the default settings	2-155
RESTART	Restarts a RESTARTABLE DUP operation	2-157
REVOKE (Super ID)	Revokes a license for a privileged program file or resets security attributes of files and programs	2-159
SECURE	Sets or changes the security attributes of a file	2-161
SET	Changes file-creation default attributes	2-165
SHOW	Displays the current settings of the file-creation attributes	2-180
STATUS	Reports the status of a reload operation	2-183
SUBVOLS	Displays the names of all subvolumes on a volume	2-185
SUSPEND	Stops a reload operation	2-186
SYSTEM	Sets the default node	2-187
TRUST	Sets or resets the Trust flag	2-188
VOLS	Displays information about volumes on a system	2-190
VOLUME	Changes the FUP default volume or subvolume names	2-190

!

Executes an existing command again.

```
! [ -num | num | string | "quoted" ]
```

-num

executes a command that appears before the current command. For example, use !-3 to execute the third command prior to the current one.

num

is the number of a command line. For example, use !2 to execute the second command of the current FUP session.

string

is the first character or characters of a previous command. For example, use !DUP $ to execute the most recent DUP command that begins with a volume name.

quoted

is a string enclosed in either single or double quotation marks. FUP searches every character in the command buffer—not just the first characters—until it finds the string. For example, use !“\MAUI” to execute the most recent command that referenced the system \MAUI.

! Guidelines

- If you use the ! command without a number or text string, FUP executes the last command you entered again.
- FUP displays the specified command before it is executed.
- To display command line numbers or recent commands, use the FUP HISTORY command.

! Examples

- To immediately view the FUP command entered on line eight and execute it again:

  ```
  -!8
  ```

- To execute the command issued two commands prior to the current command:

  ```
  -!-2
  ```

- To execute the last FUP INFO command again:

  ```
  -!INFO
  ```

Commands Related to !

COMMAND	Function	Page
HISTORY	Displays previous FUP commands	2-79
FC	Modifies a previous command	2-70
?	Displays a previous FUP command	2-5

?

Displays a specific command.

```
? [ -num | num | string | "quoted" ]
```

-num

displays a command that appears before the current command. For example, use ?-3 to display the third command prior to the current one.

num

is the number of a command line. For example, use ?2 to display the second command of the current FUP session.

string

is the first character or characters of a previous command. For example, use ?DUP $ to display the most recent DUP command that begins with a volume name.

quoted

is a string enclosed in either single or double quotation marks. FUP searches every character in the command buffer—not just the first characters—until it finds the string. For example, use ?“\MAUI” to display the most recent command that referenced the system \MAUI.

? Guidelines

- To display the last command you entered, use the ? command without a number or text string.
- To display command line numbers of recent commands, use the FUP HISTORY command.

Commands Related to ?

COMMAND	Function	Page
HISTORY	Displays previous FUP commands	2-79
FC	Modifies a previous FUP command	2-70
!	Reexecutes a previous FUP command	2-4

ALLOCATE

Allocates file extents for a disk file. This command applies only to Enscribe files.

```
ALLOCATE fileset-list , num-extents [ , PARTONLY ]
```

fileset-list

is a list of disk files for which extents are to be allocated. You can specify *qualified-fileset* for this *fileset-list*.

num-extents

is the total number of extents to be allocated to the file.

For nonpartitioned Disk Process 2 (DP2) files and key-sequenced partitioned files, specify *num-extents* as a value from 1 through *maximum-extents*. *maximum-extents* is the number of extents set with the MAXEXTENTS file attribute when the file was created or last altered. The default value for *maximum-extents* is 16. For more information, see SET on page 2-165.

For partitioned files that are not key-sequenced, specify *num-extents* as a value from 1 through 16 multiplied by the number of partitions. Each partition is allocated 16 extents (beginning with the start of the file) until the total *num-extents* are allocated.

Note. Extents 0 through 15 are in partition zero, extents 16 through 31 are in partition one, extents 32 through 47 are in partition two, and so on.

PARTONLY

allocates extents to any primary and secondary partitions of partitioned files in *fileset-list*. If a primary partition name is referenced, the extents are allocated only to the primary partition. If you omit PARTONLY, FUP allocates extents to all the partitioned files in *fileset-list*. PARTONLY has no effect on nonpartitioned files.

ALLOCATE Guidelines

- *num-extents* has a different significance for different file types. For key-sequenced partition files, it is the number of extents allocated in each partition.

For nonpartitioned files and partitioned files that are not key sequenced, it is the total number of extents to be allocated to the file.

An example is trying to allocate 12 additional extents to a nonpartitioned file that already has four extents allocated and trying to allocate 12 additional extents for the primary partition of a key-sequenced file that already has four extents allocated. Although you need to specify 16 for *num-extents* in both instances, you must include the PARTONLY option with the key-sequenced file specification so that the 12 extents are allocated only to the primary partition.

Note. You can allocate more than 16 extents on the last partition only when you use the PARTONLY option with the ALLOCATE command.

- The ALLOCATE command cannot handle SQL files that are not SQL object files. You must use the SQLCI CREATE and SQLCI ALTER commands.
- To allocate volume directory extents for *fileset-list*, you must use this file name syntax:

  $*volume*.SYS00.DIRECTRY

ALLOCATE Examples

To create an unstructured file and allocate 10 file extents for it:

1. To create an unstructured file when the default file-creation attributes are enabled:

   ```
   -CREATE YRFILE
   ```

2. Use the FUP INFO, DETAIL command to see that no file extents are initially allocated for the file:

   ```
   -INFO YRFILE, DETAIL
   $BOOKS1.COMLANG.YRFILE               14 April 2001, 9:00
        ENSCRIBE
        TYPE U
        FORMAT 1
        EXT ( 2 PAGES, 2 PAGES )
        MAXEXTENTS  16
        BUFFERSIZE  4096
        OWNER 8,44
        SECURITY (RWEP): NUNU
        DATA MODIF: 14 April 2001, 08:59
        CREATION DATE:  14 April 2001, 08:59
        LAST OPEN:  NEVER OPENED
        EOF 0  (0.0 % USED)
        FILE LABEL: 214  (5.2 % USED)
        EXTENTS ALLOCATED: 0
   ```

   This display indicates it is a DP2 file because:

   - The listing includes the MAXEXTENTS, BUFFERSIZE, CREATION DATE, and LAST OPEN attributes.

- The extent size for YRFILE is not the one page (2,048 bytes) FUP default.

  During the file-creation process, DP2 rounds up the extent size (to 2 pages or 4,096 bytes) because the extent size of DP2 files must always be an integral multiple of the BUFFERSIZE (for unstructured files) or of the BLOCK size (for structured files).

To create an unstructured DP2 file with one-page extents, you must specify a BUFFERSIZE of 2048 bytes with either the FUP SET or FUP CREATE command.

3. To allocate 10 file extents for YRFILE:

```
-ALLOCATE YRFILE, 10
```

4. Use the FUP INFO YRFILE, DETAIL command to see that the extents are now allocated:

```
$BOOKS1.COMLANG.YRFILE                14 April 2001, 09:05
     ENSCRIBE
     TYPE U
     FORMAT 1
     EXT ( 2 PAGES, 2 PAGES )
     MAXEXTENTS  16
     BUFFERSIZE  4096
     OWNER 8,44
     SECURITY (RWEP): NUNU
     DATA MODIF: 14 April 2001, 08:59
     CREATION DATE: 14 April 2001, 08:59
     LAST OPEN:  14 April 2001, 09:04
     EOF 0  (0.0 % USED)
     FILE LABEL: 214  (5.2 % USED)
     EXTENTS ALLOCATED: 10
```

Commands Related to ALLOCATE

COMMAND	Function	Page
DEALLOCATE	Deallocates unused file extents	2-60

ALLOW

Sets the number of errors and warnings that FUP allows before it stops executing FUP commands. If this number is exceeded, the current FUP command is aborted.

If you reach the error or warning limit while entering FUP commands interactively, FUP terminates the current command, displays a warning message, displays its hyphen prompt, and continues to accept commands.

If you reach the error or warning limit while FUP is executing commands from a command file, FUP stops executing.

The ALLOW command became an option of the CONFIG[URE] command with the D30 product version of FUP. However, for compatibility purposes, any FUP product versions prior to D30 continue to recognize the ALLOW option as command syntax. For more information, see CONFIG[URE] on page 2-26.

ALTER

Changes some characteristics of an Enscribe disk file's label. This command affects only the file label. It does not create or purge files, and does not insert, delete, or move records. It applies only to Enscribe files.

```
ALTER filename { , alter-option }...
```

filename

is the name of the file that you want to alter. FUP expands a partial file name by adding the current default names for system, volume, and subvolume. You cannot use wild-card characters in *filename* or specify *qualified-fileset* for it.

alter-option

names the file characteristic you want altered. Available options depend on the file type:

For all file types:

```
[ NO ] AUDIT
[ NO ] AUDITCOMPRESS
[ NO ] BUFFERED
CODE file-code
LOCKLENGTH generic-lock-key-length
MAXEXTENTS maximum-extents
NOPURGEUNTIL timestamp
[NO] REFRESH
RESETBROKEN
RESETCORRUPT
[ NO ] SERIALWRITES
[ NO ] VERIFIEDWRITES
```

For files with alternate-key fields:

```
ALTFILE ( key-file-number , filename )
ALTKEY ( key-specifier { , altkey-param }... )
DELALTFILE key-file-number
DELALTKEY key-specifier
```

For partitioned files:

```
PART ( sec-partition-num ,
     [ \node.]$volume
     [ , pri-extent-size [ , sec-extent-size ]] )
PARTONLY
```

For unstructured files:

```
BUFFERSIZE unstructured-buffer-size
ODDUNSTR
```

ALTER Parameters for All File Types

These *alter-option* parameters are available for all file types:

`[ NO ] AUDIT`

designates the file as audited or nonaudited by TMF. If you specify AUDIT, FUP marks the file as an audited file. If you specify NO AUDIT, FUP marks the file as a nonaudited file. To use the AUDIT option, you must have read and write access to a file (or be the super ID 255,255).

Note. For more information, see the *TMF Planning and Configuration Guide.*

`[ NO ] AUDITCOMPRESS`

sets the mode of producing audit-checkpoint messages (using compressed or entire messages) for audited files. The default is NO AUDITCOMPRESS.

`[ NO ] BUFFERED`

sets the mode of handling write requests to the file using buffered or write-through cache. BUFFERED specifies a buffered cache. NO BUFFERED specifies a write-through cache. The default is BUFFERED for audited files and NO BUFFERED for nonaudited files.

△ **Caution.** If you use the buffered-cache option on a DP2 file that TMF does not audit, a system failure or disk-process takeover can cause the loss of buffered updates to the file. An application might not detect this loss, or handle the loss correctly unless it is modified to do so.

`CODE` *`file-code`*

alters the file code. Specify *`file-code`* as an integer from 0 through 65,535. The default *`file-code`* is zero. File codes 100 through 999 are reserved for use by HP.

Note. For a list of reserved file codes, see Table 2-2, System File Code Definitions, on page 2-87.

`LOCKLENGTH` *`generic-lock-key-length`*

is the lock-key length. The generic lock-key length determines the grouping of records that share a single lock. The value specified must be between zero and the key length of the file.

`MAXEXTENTS` *`maximum-extents`*

sets the maximum number of extents to allocate (for nonpartitioned files only). Specify *`maximum-extents`* as an integer from 16 through *`n`*, where *`n`* is a maximum value determined by the amount of free space in the file label. FUP rounds any value you set from 1 through 15 up to 16. The absolute maximum is 978 extents. The default is 16.

You cannot always allocate all the extents specified by *`maximum-extents`*. The actual number of extents that you can allocate depends on the amount of space in the file label. If there are alternate keys or partitions, the maximum number of extents allowed is fewer than 978.

If you specify MAXEXTENTS and the file is Format 1, you must consider the primary and secondary extent sizes to avoid exceeding the maximum file size for a Format 1 file. When the primary and secondary extent size with the specified MAXEXTENT size is larger than two gigabytes, the ALTER operation rejects the request with file-system error 21 (illegal count specified) if the file is Format 1.

When the primary and secondary extent size with the specified MAXEXTENT size is larger than four gigabytes, the ALTER operation rejects the request with file-system error 12 (file in use) if the file is:

- format 2 file that is not key-sequenced and not partitioned
- format 2 partitioned file that is key-sequenced and the operation is performed on other than the base partition
- currently less than 4 gigabytes
- currently opened by another process

If the file is a partitioned file that is not key-sequenced, the ALTER operation rejects the request with file-system error 2 (operation not allowed on this type of file).

NOPURGEUNTIL [*timestamp*]

lets you change the expiration date of a file.

timestamp is:

[*date*] [,] [*time*]

where *date* is specified as:

{ *dd mmm yyyy* | *mmm dd yyyy* }

where *dd* (day) is an integer from 1 through 31, and *mmm* (month) is one of:

```
JAN, FEB, MAR, APR, MAY, JUN,
JUL, AUG, SEP, OCT, NOV, DEC
```

and *yyyy* (year) is a four-digit integer in the range 1900 to 3999.

If *date* is omitted, *timestamp* defaults to the current date.

time is specified as:

hh:*mm*[:*ss*]

values for the hour (*hh*) are from 0 to 23 (only the 24-hour clock is supported).

If *time* is omitted, *timestamp* defaults to the beginning of the day (0:00:00 or midnight).

If seconds (*ss*) is omitted, *timestamp* defaults to zero.

If both *date* and *time* are omitted, *timestamp* returns to the parameter default setting at the time the file was created.

If *timestamp* contains a date prior to Jan 1, 1900 or after Dec 31, 4000, FUP displays BAD TIMESTAMP and no date in the INFO DETAIL response screen.

[NO] REFRESH

causes the file label to be copied to disk whenever the file control block is marked as dirty. This situation occurs if the end of file or a free-space block changes. The file label is not written to disk if the only change is updating the LAST MODIFIED field. The default is NO REFRESH. (The label is not copied to disk.)

RESETBROKEN

resets the BROKEN flag in the file label for a nonaudited file. Fix the broken parts of the file before using RESETBROKEN.

RESETCORRUPT

causes the corrupt flag to be reset (turned off).

`[ NO ] SERIALWRITES`

sets the mode of writes to the mirror: serial or parallel. The default is NO SERIALWRITES.

`[ NO ] VERIFIEDWRITES`

sets the mode of file writes: verified or unverified. The default is NO VERIFIEDWRITES.

ALTER Parameters for Files With Alternate-Key Fields

These *alter-option* parameters are available for files with alternate-key fields:

`ALTFILE ( key-file-number, filename )`

adds or replaces the file name of an alternate-key file. You must include this parameter for any undefined key file number referenced by an ALTKEY specification. To designate the alternate-key file being named, specify *key-file-number* as an integer in the range 0 through 255.

The FUP process expands a partial file name by adding the current default names for node, volume, and subvolume.

If you add a new ALTFILE number, your ALTER command must also include an ALTKEY option that specifies the alternate-key file.

If you define a new alternate-key file, you must create the file in a separate operation either before or after using ALTER ALTFILE.

If you delete the last alternate-key specification, you must also delete the corresponding alternate-file specification in the same ALTER command.

If you delete the last alternate-file specification, you must also delete the corresponding alternate-key specification in the same ALTER command.

`ALTKEY ( key-specifier { , altkey-param }... )`

adds, replaces, or alters an alternate-key specification.

key-specifier

is a 2-byte value that identifies this alternate-key field. Specify *key-specifier* as either a one or two-character string in quotation marks:

"[*c1*]*c2*"

or as an integer from -32,768 through 32,767:

`{ -32,768 : 32,767 }`

You can use any characters for *key-specifier* except zero. If you omit *c1*, then *c1* is treated as a zero.

Note. If you add a new key specifier that references an undefined key-file number, you must include the ALTFILE option to define the alternate-key file.

---

altkey-param

specifies attributes of the alternate-key file.

FILE *key-file-number*

sets the key-file number for *key-specifier*. Specify *key-file-number* as an integer in the range 0 through 255. This number is related to an actual file by the ALTFILE *create-param*. The default *key-file-number* is zero.

[NO] INSERTIONORDER

specifies whether to use insertion-ordered alternate-key sequencing. The default, NO INSERTIONORDER, specifies to order alternate key records of files with duplicate key values by their primary key sequence and not their order of insertion.

An insertion-ordered alternate key cannot share an alternate-key file with other keys of different lengths or with other keys that are not insertion ordered.

All the nonunique alternate keys of a file must have the same duplicate-key ordering attribute. A file with this specification must not have both insertion-ordered alternate keys and standard (duplicate ordering by primary key) nonunique alternate keys.

KEYLEN *key-length*

sets the key length for *key-specifier*. To create a key-sequenced file, you must specify a KEYLEN, or the creation attempt fails.

KEYOFF *key-offset*

sets the key offset for *key-specifier*. The default is zero.

NO NULL | NULL *null-value*

specifies whether to set a null value for *key-specifier*. If a value is specified, *null-value* must be an ASCII character in quotation marks (or an integer in the range 0 through 255). The default is NO NULL.

Note. For more information about null values, see the *Enscribe Programmer's Guide.*

`[ NO ] UNIQUE`

specifies whether to set *key-specifier* as a unique key. The default is NO UNIQUE.

`[ NO ] UPDATE`

specifies whether to set automatic updating for the alternate-key file represented by *key-specifier*.

The NO UPDATE option prevents the file system from automatically updating the specified alternate-key file when you write to the main file. Although you usually want to keep alternate-key files synchronized with their main files, you sometimes might want to keep files unsynchronized. For example, you might have two files pointing to the same alternate-key file but only want updates from one of the two written to it.

The default is UPDATE.

`DELALTFILE` *key-file-number*

deletes the reference to an alternate-key file but does not purge the file. The alternate-key file must not be referenced by any existing *key-specifier*. After you execute an ALTER command with this option, the remaining key-file numbers and references to them are adjusted. The numbers begin with zero and are contiguous (0,1, 2, and so on).

`DELALTKEY` *key-specifier*

deletes an alternate-key specification. You cannot access the file through *key-specifier* after you execute this option. The *key-specifier* parameter is a 2-byte value that you already specified (in the ALTKEY parameter) to identify the alternate-key field. The value is passed to the KEYPOSITION procedure when it is referenced by this key field.

Specify *key-specifier* as a one or two-character string in quotation marks:

"[*c1*]*c2*"

Or specify it as an integer from -32,768 through 32,767:

{ -32,768 : 32,767 }

You can use any characters for *key-specifier* except zero. If you omit *c1*, then *c1* is treated as a zero.

ALTER Parameters for Partitioned Files

These *alter-option* parameters are available for partitioned files:

```
PART ( sec-partition-num , [\node.]$volume
     [ , pri-extent-size [ , sec-extent-size ] ] )
```

PART

alters secondary partition specifications for partitioned files. You can specify extent sizes if you are adding a new partition or if you are altering an existing partition of a key-sequenced file. You must specify each secondary partition separately.

If the *sec-partition-num* already exists, for all file types, you can change *volume*.

You can redefine the extent sizes for key-sequenced files only. If the partition contains data, you must perform additional operations on the partition to complete the extent-size change.

You can add a new, additional partition to any files that are not key sequenced. You cannot add partitions to key-sequenced files.

sec-partition-num , \node.$volume

names the volume where this secondary partition is to reside. Specify *sec-partition-num* as an integer from 1 through 15 to designate the secondary partition. Specify *node* and *volume* as the names of the node and volume to contain the partition. The file name and the subvolume of the primary partition are specified when the file is created.

Although FUP lets you specify any number in the range 1 through 15 for *sec-partition-num*, FUP changes it to a standard DP2 number that starts at zero when the file is created.

Note. Example 2-6 on page 2-98 shows how the DP2 number is listed in the INFO DETAIL command.

---

pri-extent-size , sec-extent-size

alters the primary and secondary extent sizes. The default value for *pri-extent-size* is one page (2,048 bytes). If you specify a value of zero or do not specify a value, *sec-extent-size* defaults to the *pri-extent-size* value. You cannot define primary or secondary extents as zero pages when altering files.

For partitioned unstructured files where one or more partitions reside on a disk drive in a disk drive enclosure, DP2 has additional restrictions. You must specify both the *pri-extent-size* and the *sec-extent-size* so that they can be explicitly divisible by 14. DP2 does not automatically round the size up. Additionally, you must specify the same *pri-extent-size* for all partitions and the same *sec-extent-size* for all partitions.

You can specify these values for *pri-extent-size* and *sec-extent-size*:

`0:`*`maximum`* `[ PAGE[S] ]`

specifies the extent size in pages (2,048-byte units). Possible values are:

Format 1	Format 2
`0:65,535 [ PAGE[S] ]`	`0:512,000,000 [ PAGE[S] ]`

Note. If you specify an extent size over 65,535 pages, you must assign Format 2 to your files. For more information about Format 2 files, see Handling Different Types of Files on page 1-22.

Because the minimum extent size is one page (2,048 bytes), one page is also allocated if you specify zero extents.

`0:`*`maximum`* `BYTE[S]`

specifies the extent size in bytes. Possible values of *`maximum`* are:

Format 1	Format 2
`0:134,215,680 BYTE[S]`	`0:2,147,483,647 BYTE[S]`

The FUP process rounds the extent size up to the next full page. If you specify 2,047 bytes, FUP allocates one page. If you specify 2,049 bytes, FUP allocates two pages.

`0:`*`maximum`* `REC[S]`

specifies the extent size based on the current settings for *`record-length`* (REC), *`data-block-length`* (BLOCK), *`index-block-length`* (IBLOCK), key-field lengths, and compression settings. Possible values are:

Format 1	Format 2
`0:134,215,680 REC[S]`	`0:2,147,483,647 REC[S]`

The FUP process rounds the extent size up to the next full page.

`0:`*`maximum`* `MEGABYTE[S]`

specifies extent sizes in million-byte units. Possible values are:

Format 1	Format 2
`0:134 MEGABYTE[S]`	`0:2147 MEGABYTE[S]`

The FUP process rounds the extent size up to the next full page.

`PARTONLY`

specifies that any changes you make with the ALTER command apply only to the indicated file partition. If you reference a primary partition name, the extents are altered only in the primary partition.

ALTER Parameter for Unstructured Files

These *alter-option* parameters are available for unstructured files:

BUFFERSIZE *unstructured-buffer-size*

is the internal buffer size to use when accessing the specified file. You can set the BUFFERSIZE file attribute with this command (FUP ALTER) or with the FUP CREATE and FUP SET commands. Possible values for *unstructured-buffer-size* (in bytes) are:

```
512, 1024, 2048, 4096
```

FUP rounds the actual buffer size up to the nearest valid DP2 block size.

ODDUNSTR

changes an even unstructured file to an odd unstructured file.

Unstructured Enscribe files can be even or odd. FUP rounds up any odd byte count that you give to an even unstructured file (for reading, writing, or positioning). This is the default for unstructured files.

FUP does not round odd unstructured files up. You always read, write, or position at the byte count you give.

To change an odd unstructured file to an even unstructured file, copy the odd file into a new file that was created as even unstructured.

ALTER Guidelines

- To receive the current file attributes for any file you want to alter, use the FUP INFO command:

  -INFO *filename* , DETAIL

- To alter a file, you must have both read and write access to it.
- Changing the AUDIT option for DP2 files also changes the default value of the BUFFERED attribute.

  If you specify NO AUDIT, the BUFFERED option and file label default is set to NO BUFFERED. If you specify AUDIT, the BUFFERED option is set ON. If you have explicitly set the BUFFERED attribute, that value remains unchanged.

- Adding the AUDIT attribute to a file causes audit records to be written for the file.
- If you use the AUDIT option and the volume containing the primary file, or any of its secondary partitions or alternate-key files containing at least one automatically updated alternate key, is not audited, the request fails. You receive file-system error 80 (invalid operation on audited file or nonaudited disk volume). If these files are all audited, the labels of the alternate-key files are updated to reflect the audit option.

- To add an alternate key to a structured file that does not have any alternate-key files specified, your ALTER command must specify both the new alternate key and the new alternate-key file. This is also true if you try to delete an alternate key with the DELALTFILE option of the ALTER command.

  For example, to add the alternate key “aa” to the file FRED and specify the alternate-key file AFILE:

  ```
  -ALTER FRED, ALTFILE (0, AFILE), ALTKEY ("aa", FILE 0,&
  -KEYOFF 0, KEYLEN 5)
  ```

- Altering the NULL or UPDATE attribute of an alternate key for a file does not change the actual contents of the alternate-key file. You must update the data (usually with the FUP LOADALTFILE command) to make the alternate-key file completely consistent with the primary-key file. If your application does not require complete consistency, you might not need to reload the alternate-key file.

- Altering the UNIQUE attribute of an alternate key for a file makes the file description inconsistent with its alternate-key files. Subsequent attempts to open the file cause file-system error 4 (failure to open an alternate-key file). To use the file after the UNIQUE attribute for an alternate key is altered, purge the alternate-key file, re-create it, and adjust its key length. Files with the UNIQUE attribute have a different key length than files with the NO UNIQUE attribute.

Note. For more information about the storage of alternate keys in alternate-key files, see the *Enscribe Programmer’s Guide.*

- When you alter an alternate key from NO UNIQUE to UNIQUE, the contents of the records in the primary file are not examined. The file could contain records with duplicate values in the alternate-key field while you are changing its description.

  The FUP ALTER command does not recognize the existence of duplicates, but the duplicates cause an error when the alternate-key file is loaded.

- If the extent sizes of a secondary partition are altered, this warning message appears:

  ```
  USING SPECIFIED EXTENT SIZES:  USER MUST ENSURE
  CONSISTENT PARTITIONS
  ```

- The ALTER command cannot handle SQL files that are not SQL object files. To do this, you must use SQLCI TABLE and SQLCI INDEX.

- The ALTER command returns error 197 (an SQL error has occurred) against SQL-compiled objects.

- All partitions of a file are created with the same format version. For files that are not key-sequenced, a partition created independently must have the same format as all other partitions of the file.

△ **Caution.** If you use the ALTER command to change a partition to a different format from the rest of the partitions, the file system reports errors. FUP does not check for such errors.

ALTER Examples

- To assign file code 10 to MYFILE1:

  ```
  -ALTER MYFILE1, CODE 10
  ```

- To cause the file label for MYFILE2 to be updated whenever the file control block (FCB) changes:

  ```
  -ALTER MYFILE2, REFRESH
  ```

- To assign the alternate-key file MYFILE4 to MYFILE3 and give it key-file number two (ALTFILE 2 must already be a defined attribute of MYFILE3):

  ```
  -ALTER MYFILE3, ALTFILE (2 , MYFILE4)
  ```

- To delete the alternate key "ab" from MYFILE5:

  ```
  -ALTER MYFILE5, DELALTKEY "ab", DELALTFILE 0
  ```

Commands Related to ALTER

COMMAND	Function	Page
ALLOCATE	Allocate extents for a file	2-6
GIVE	Change ownership of a file	2-76
SECURE	Change security, PROGID, or CLEARONPURGE attributes of a file	2-161

BUILDKEYRECORDS

Generates the alternate-key records for specified key fields of a structured disk file and writes those records to a designated file. This command applies only to Enscribe files.

Although the output of BUILDKEYRECORDS can be the actual destination alternate-key file, it is more efficient to use a LOADALTFILE command to load alternate-keys. You can use BUILDKEYRECORDS to generate the alternate-key records, store them in another file, and then load them into the destination alternate-key file with a LOAD command. This approach is useful when limited system resources prevent a LOADALTFILE operation.

```
BUILDKEYRECORDS primary-filename , out-filename
              , key-specifier-list [ , out-option ] ...


key-specifier-list is:

   key-specifier
   ( key-specifier [ , key-specifier ] )


out-option is:

   BLOCKOUT out-block-length
   DENSITYOUT density
   EBCDICOUT
   PAD [ pad-character ]
   RECOUT out-record-length
   [ NO ] REWINDOUT
   SKIPOUT num-eofs
   TAPEMODE mode
   [ NO ] UNLOADOUT
   XLATE [ translation-table-name ]
   XLATEIN [ translation-table-name ]
   XLATEOUT [ translation-table-name ]
```

primary-filename

names an existing primary file whose alternate-key records are to be generated. You must have previously defined alternate-key fields for the primary file. You cannot use wild-card characters or specify *qualified-fileset* for *primary-filename*.

out-filename

names an existing file where the alternate-key records generated by this command are to be written. *out-filename* can be an existing disk file, a nondisk device, a process, a tape or SPOOLER (code 129) file, or a spool DEFINE name. You cannot use wild-card characters or specify *qualified-fileset* for *out-filename*.

If *out-filename* (the destination file) contains existing data, it is never overwritten during a BUILDKEYRECORDS operation.

key-specifier-list

names one or more alternate-key fields of the primary file whose corresponding alternate-key records are to be generated.

key-specifier

is a 2-byte value that identifies the alternate-key field. Specify it as a one or two-character string in quotation marks:

"[*c1*]*c2*"

Or as an integer from -32,768 through 32,767:

```
{ -32,768 : 32,767 }
```

You can use any characters for *key-specifier* except zero. If you omit *c1*, then *c1* is treated as a zero.

out-option

specifies the format and control of *out-filename*. *out-option* is any one of:

```
BLOCKOUT out-block-length
DENSITYOUT density
EBCDICOUT
PAD [ pad-character ]
RECOUT out-record-length
[ NO ] REWINDOUT
SKIPOUT num-eofs
TAPEMODE mode
[ NO ] UNLOADOUT
XLATE [ xlate-table-name ]
XLATEIN [ xlate-table-name ]
XLATEOUT [ xlate-table-name ]
```

Note. For a complete description of these options, see out-option on page 2-44.

BUILDKEYRECORDS Guidelines

- BUILDKEYRECORDS causes the primary file to be read sequentially according to its primary-key field.

  For each record read from the primary file, BUILDKEYRECORDS generates one or more alternate-key records (corresponding to the number of key specifiers that are named) and writes them to the out file.

  If you name more than one key specifier, BUILDKEYRECORDS generates the corresponding alternate-key file records in the order of the ASCII collating sequence for the key specifiers.

- BUILDKEYRECORDS honors any NULL specification defined for a key field. Alternate-key records are not generated for any fields that consist only of a null character.
- BUILDKEYRECORDS ignores NO UPDATE specifications.
- BUILDKEYRECORDS ignores UNIQUE specifications. Duplicate unique-key values are detected when the LOADALTFILE command loads the alternate-key file.
- When alternate key records are not built because the full alternate key does not exist within the primary record, this message appears:

  nnn `RECORDS CONTAIN INCOMPLETE ALTERNATE KEY FIELDS (ALTERNATE KEY RECORDS NOT GENERATED)`

- BUILDKEYRECORDS cannot handle SQL files.

BUILDKEYRECORDS Example

To generate the alternate-key records for MYFILE using key specifications “ab” and “cd”, and write the alternate-key records to $TAPE:

```
-BUILDKEYRECORDS MYFILE, $TAPE, ("ab","cd")
```

Commands Related to BUILDKEYRECORDS

COMMAND	Function	Page
LOADALTFILE	Creates an alternate-key file from a primary file	2-136
LOAD	Creates a structured file from scratch	2-129
CONFIG[URE]	Sets default options for the BUILDKEYRECORDS command	2-26

CHECKSUM

Recomputes the checksum value for blocks of data in disk files. Use this command when recovering from a software-detected checksum error.

```
CHECKSUM fileset-list [ , PARTONLY ]
```

fileset-list

is a list of files whose checksum values are to be recomputed. *fileset-list* can include Enscribe files (structured or unstructured) and SQL files. You can specify *qualified-fileset* for *fileset-list*.

Any SQL views (protection or shorthand) in *fileset-list* are ignored, and a warning is issued.

PARTONLY

specifies that FUP is to compute checksum values for all the primary and secondary partitions encountered in *fileset-list* and for all nonpartitioned files. If a primary partition name is referenced, the secondary partitions of the file are not checksummed.

CHECKSUM Guidelines

- Although CHECKSUM reads each block of data (from each file specified by *fileset-list*) and recomputes a checksum value for each one, it rewrites only blocks whose checksum values are incorrect.
- Checksum errors usually indicate a potential data integrity problem. CHECKSUM recomputes the checksum value for blocks of data but does not fix any data that might have changed.
- If a Peripheral Utility Program (PUP) DOWN or STOPOPENS command was executed for the volume named in *fileset-list*, CHECKSUM aborts.
- If an open file or SQL view is encountered during processing, CHECKSUM skips it, and a warning message appears.
- If the *fileset-list* includes an asterisk (*) to specify all files in the file set, and if the PARTONLY option has not been specified, CHECKSUM skips secondary partitions of partitioned files (Enscribe and SQL).
- CHECKSUM accepts SQL catalogs but not SQL views.
- The maximum number of files in the *fileset-list* that can be checksummed at a time is 2,147,483,647.

CHECKSUM Examples

- To recompute checksum values for all the files on volume $TEST1 (except secondary partitions of partitioned files):

  ```
  -CHECKSUM $TEST1.*.*
  ```

  If partitioned files exist on $TEST1, FUP skips the secondary partitions because the wild-card character (*) is specified for the files in *fileset-list*, and the PARTONLY option is not included in the command.

- To recompute checksum values for all the files on volume $TEST1 (including any existing partitioned files):

  ```
  -CHECKSUM $TEST1.*.*, PARTONLY
  ```

- In this example, assume that a checksum error for $DATA appears on the operator console and that $DATA contains these SQL and Enscribe files:

  - An SQL table named KSTABLE
  - An index on KSTABLE named KSTABLEI
  - An SQL protection view named PVIEW that depends on table KSTABLE
  - An SQL shorthand view named SVIEW that also depends on KSTABLE
  - An SQL program file named OFILE
  - A structured Enscribe file named SFILE
  - An unstructured Enscribe file named UFILE

  To recompute checksum values for all the files on volume $DATA:

  ```
  10> FUP CHECKSUM $DATA.*.*
  ```

  This command produces these results:

  - Checksum values are recomputed for SQL files KSTABLE and KSTABLEI, for the SQL object program file OFILE, and for the Enscribe files SFILE and UFILE.
  - SQL views PVIEW and SVIEW are skipped, and a warning message appears.

CONFIG[URE]

Customizes your FUP configuration information. You can use the CONFIG abbreviation for this command. The COPY, DUP[LICATE], LOAD, LOADALTFILE, and RELOAD configuration commands modify the defaults of their corresponding FUP commands. These modified defaults become applicable when the corresponding command is executed. The commands apply to other FUP commands as specified.

```
CONFIG[URE] config-command [ config-params ]...


config-command is:

command-option | environment-option

  command-option is:

    COPY copy-option [, copy-option ]...
    DUP[LICATE] dup-option [, dup-option ]...
    LOAD load-option [, load-option ]...
    LOADALTFILE loadaltfile-option [, loadaltfile-option ]...
    RELOAD reload-option [, reload-option ]...

  environment-option is:

    ALLOW allow-option [, allow-option ]...
    DISPLAYBITS bitcount
    [ NO ] ECHO [ CONFIG[URE] ] [ OBEY ]
    IOTIMEOUT time
    NETBLOCKSIZE size
    [ NO ] PROMPT [ PURGE ]
    REPORTWIDTH width
    RESTARTUPDATE time
    STATONLY
    XLATE xlate-table-name  [ TEXT|CHARMAP ]
                            [ IN filename  ]
```

config-command

specifies FUP configuration information:

`COPY` *copy-option* `[,` *copy-option* `]...`

specifies any of the COPY options (except SOURCE or TARGET file names); for example, EBCDICIN, BLOCKIN 4096. By default, no COPY options are configured. For more information, see COPY: Copy Form on page 2-35.

`DUP[LICATE]` *dup-option* `[,` *dup-option* `]...`

specifies any of the DUP[LICATE] options (except SOURCE or TARGET file names); for example, PURGE, SOURCEDATE. By default, no DUP[LICATE] options are configured. For more information, see DUP[LICATE] on page 2-61.

`LOAD` *`load-option`* `[,` *`load-option`* `]...`

specifies any LOAD option (except SOURCE or TARGET file names); for example, SLACK 20. By default, no LOAD options are configured. For more information, see LOAD on page 2-129.

`LOADALTFILE` *`loadaltfile-option`* `[,` *`loadaltfile-option`* `]...`

specifies any of the LOADALTFILE options (except *`key-file-number`* and *`primary-filename`*); for example, MAX 999, SLACK 50. By default, no LOADALTFILE options are configured. For more information, see LOADALTFILE on page 2-136.

`RELOAD` *`reload-option`* `[,` *`reload-option`* `]...`

specifies any of the RELOAD options (except *`filename`*); for example, NEW, DSLACK 60. By default, no RELOAD options are configured. For more information, see RELOAD on page 2-146.

`ALLOW` *`allow-option`* `[,` *`allow-option`* `]...`

specifies the number of errors FUP permits while processing commands. The default is zero errors. (This option replaces the FUP command ALLOW on page 2-9.)

`allow-option` is:

```
{ num } [ [SEVERE ] ERRORS ]
{ num } [ WARNINGS         ]
[ ABENDS [ ON | OFF ]      ]
```

`num`

is the maximum number of errors or warnings that can occur before a FUP command is aborted. The default ALLOW count is 0 for severe and nonsevere errors; it is unlimited for warnings. The allowable range for errors is 0 through 32767. FUP keeps an error count internally for both severe and nonsevere errors.

`num` `ERRORS`

sets the nonsevere error count to *`num`*.

`num` `SEVERE ERRORS`

sets both the severe and nonsevere error counts to *`num`*.

`num` `WARNINGS`

sets the number of warnings that FUP allows while executing FUP commands.

```
ABENDS [ ON | OFF ]
```

specifies whether FUP should terminate abnormally (ABEND) when it encounters an error and the allowed count for the error was exceeded. The default for ABENDS depends on:

- The default when FUP is not used interactively is ON.
- If FUP is used interactively (the IN file is a terminal), the ABENDS option is set to OFF.
- If ABENDS is specified without ON or OFF, the ABENDS option is set to ON.
- Errors from which the FUP process cannot recover always cause FUP to terminate abnormally.

Note. Table 2-1 shows FUP responses to warnings and error types when ALLOW ABENDS is ON or OFF.

Table 2-1. Response to ALLOW ABENDS ON or OFF

Type of Error	ALLOW ABENDS OFF	ALLOW ABENDS ON
Warning	Warning allow count is decremented. If allow count > 0, the command continues. If allow count <= 0, the command fails.	Warning allow count is decremented. If allow count > 0, the command continues. If allow count <= 0, FUP terminates abnormally.
Nonsevere error	Nonsevere error allow count is decremented. If allow count > 0, FUP continues with the current command on the next file in the file set. If allow count <= 0, the command fails.	Nonsevere error allow count is decremented. If allow count > 0, FUP continues with the current command on the next file in the file set. If allow count <= 0, FUP terminates abnormally.
Severe error	Severe error allow count is decremented. If allow count > 0, FUP continues with the current command on the next file in the file set. If allow count <= 0, the command fails.	Severe error allow count is decremented. If allow count > 0, FUP continues with the current command on the next file in the file set. If allow count <= 0, FUP terminates abnormally.
Syntax or semantic error	The command fails.	FUP terminates abnormally.
Nonrecoverable (calamity) error	FUP terminates abnormally.	FUP terminates abnormally.

DISPLAYBITS *bitcount*

lets the COPY, INFO, and SHOW commands display 8-bit character sets. The default is 7-bit characters.

bitcount

is the number of bits used to determine a printable character for the FUP display commands (COPY, INFO, and SHOW). Use 8 for 8-bit character sets or 7 for 7-bit character sets. The default value is 7.

[NO] ECHO [CONFIG[URE]] [OBEY]

ECHO CONFIGURE

specifies that FUP should display the options currently configured to the home terminal (or list file) when a FUP command that can be configured is executed.

If no options are configured for the current command, nothing is displayed. For example, FUP displays the current CONFIGURE options for a DUP[LICATE] process when a DUP[LICATE] command is executed.

All options configured for a specific command are displayed, including any that are explicitly overridden on the command.

ECHO OBEY

specifies that FUP should display the FUP commands that are executed in a command file to the home terminal (or list file). If you specify the ECHO OBEY option in a command file (including the FUPCSTM or FUPLOCL files), it causes an echo to start on the next command in the file.

You can specify CONFIGURE and OBEY on the same command. The ECHO option without any parameters implies both CONFIGURE and OBEY. The default is ECHO OBEY and NO ECHO CONFIGURE.

IOTIMEOUT *time*

specifies the time (in seconds) FUP should use for I/O timeouts with any DUP[LICATE] commands. Use this option only at sites where configurations can cause a DUP[LICATE] process to time out. The default values are two minutes for disk transfers and ten minutes for optical transfers.

Note. A value of 0 for *time* specifies the FUP defaults, and -1 specifies no timeouts. No other negative values are permitted.

NETBLOCKSIZE *size*

specifies the size of the blocks a DUP[LICATE] process uses to transfer data when the SOURCE or TARGET files are not on the nodes where FUP is currently running. The size is expressed in units of 1024 bytes. You must

specify the size using numbers from 4 through 28. For example, 28 specifies 28 KB block transfers. FUP rounds numbers down to the nearest 4KB units.

By default, FUP uses 28 KB transfers when the files are not local, and 56 KB transfers for local files.

`[ NO ] PROMPT [ PURGE ]`

specifies if you want to be prompted prior to any PURGE processes. The NO PROMPT PURGE option directs FUP to purge files without issuing any prompts. The PROMPT PURGE option directs FUP to issue prompts before purging files—unless an "!" was entered with the PURGE command. PROMPT without any parameters implies purge. The default is PROMPT.

`REPORTWIDTH` *`width`*

specifies the maximum length (in columns) that FUP uses to format output for commands (if applicable). *`width`* can be from 80 through 132. Although this option replaces the FUP REPORTWIDTH command, it has the same syntax and meaning.

`RESTARTUPDATE` *`time`*

specifies the time duration (in seconds) between updates to the ZZRESTRT file on any DUP[LICATE] process with the RESTARTABLE option. By default, the interval is 30 seconds for disk transfers and 10 seconds for optical transfers. The time you specify must be a positive number; 0 specifies the FUP defaults.

Because each update to the RESTART file requires a synchronized update to the TARGET file, smaller values can slow down a DUP[LICATE] process.

`STATONLY`

suppresses the implicit INFO DETAIL display if INFO STAT is specified.

`XLATE` *`xlate-table-name`* `[ TEXT|CHARMAP ] [ IN` *`filename`* `]`

specifies the name of a translation table that FUP can use in subsequent COPY, LOAD, or BUILDKEYRECORDS commands when you use the XLATE, XLATEIN, or XLATEOUT keyword in the command.

The *`xlate-table-name`* parameter is 24 characters long and can include letters, numbers, underscores (_), hyphens (-), and circumflex (^) characters.

CHARMAP specifies a translation table from the CHARMAPS product (T9279). If CHARMAP is specified, *`filename`* defaults to $SYSTEM.SYSTEM.ZCHARMAP, and *`xlate-table-name`* must be one of the defined tables in the CHARMAP file. CHARMAP is the default.

TEXT specifies a user-created translation table. If you specify TEXT, you must also specify *`filename`*. FUP expects the translation table to be the first 512

bytes of an unstructured file from the first record of a structured file. If the record is either longer or shorter than 512 bytes, FUP issues a warning message.

Because FUP does not read *filename* until needed by a COPY, LOAD, or BUILDKEYRECORDS command, the CONFIGURE command does not provide error notification.

When used in a COPY, LOAD, or BUILDKEYRECORDS command, the XLATEIN and XLATEOUT options specify direction:

XLATEIN	Specifies a translation using the first 256 bytes in the XLATE table
XLATEOUT	Specifies the second 256-byte section of the XLATE table

The XLATEOUT table is expected to be the reciprocal translation for the XLATEIN table. The XLATE option is the same as XLATEIN.

CONFIG[URE] Guidelines

- The ALLOW, DISPLAYBITS, and REPORTWIDTH commands from the previous product versions of FUP are now CONFIGURE commands. However, for compatibility, they are also supported as separate commands.
- Any values specified in the CONFIGURE command are enabled until they are reset (with a RESET CONFIGURE command) or until the FUP session ends. To save configuration values, use the SHOW command. You can set configuration values from the FUPCSTM or FUPLOCL files or from a file with an OBEY command.
- FUP uses the current configuration options wherever applicable. You can set or change the configuration options in the FUPLOCL file or the FUPCSTM file or do so interactively with the CONFIGURE or RESET commands.
- If an option is specified more than once, FUP uses the last one specified.
- Configuration options have precedence over the standard FUP defaults, but explicit options at the command line can override them—unless the configuration option does not have a named default.

  To cancel configuration options that were set with the CONFIGURE command, you must use the RESET command.

  For example, if you specify CONFIGURE COPY UNSTRUCTURED, the UNSTRUCTURED value applies to all COPY commands until it is reset. You cannot override the UNSTRUCTURED option in a COPY command because it does not have a named default. To cancel it, you must enter RESET COPY UNSTRUCTURED.

  However, the explicit SOURCEDATE option in a DUP A, B, SOURCEDATE command can override a SAVEALL option used in a CONFIGURE DUP SAVEALL command. SOURCEDATE and SAVEALL are in the same family of options, and

the explicit option in the DUP command overrides the option in the CONFIGURE command.

- Because CONFIGURE remembers options that contain file or volume names, such as DUP ALTFILE *filename* or LOAD PARTOF *volume*, you must be careful. The file names often apply only to specific parts of a file and cannot be overridden because they have no named defaults.
- Although using CONFIGURE with options that have no named defaults (including the options with file names) is more difficult than using options with named defaults, it can be useful on consecutive FUP commands that apply to the same circumstance.
- You can use the ALLOW *config-command* to force FUP to execute a command on each file within a file set when some of the commands might generate errors. The allowed error counts do not apply to command parsing errors or errors that FUP classifies as calamities (errors from which FUP cannot recover).

Note. Most file-system errors are classified as severe.

- To set defaults for COPY, INFO (DETAIL), and SHOW commands, use CONFIG[URE] DISPLAYBITS:
  - When *bitcount* is 7, COPY displays the character for any byte with an octal value from %40 through %177. A period (.) appears for an octal value greater than %177. When *bitcount* is 8, COPY displays the character for any byte with an octal value of %40 or greater.
  - When *bitcount* is 7, the ALTKEY key specifier of INFO, DETAIL, and SHOW displays for any character with an octal value greater than %177. When *bitcount* is 8, the ALTKEY key specifier is displayed for all characters with an octal value greater than %40.
  - You can switch between the 7-bit and 8-bit display modes during an interactive FUP session. To reset the display mode, enter another DISPLAYBITS command at the FUP prompt with the desired mode.
- To set defaults for COPY (with a DUMP option), FILES, SUBVOLS, and LISTLOCKS commands, use CONFIG[URE] REPORTWIDTH:
  - If the OUT file (or OUT device) record length is less than REPORTWIDTH, FUP breaks the output records into multiple records (or lines) to fit the file or device record length.
  - The output format for all other FUP commands that output information is 80 columns—regardless of the value specified for REPORTWIDTH.
- You can specify translation table defaults for COPY, LOAD, and BUILDKEYRECORDS commands:
  - If you use the XLATE option but an XLATE table was not specified in the CONFIGURE command, an error occurs.

- If you specify PAD or TRIM characters in a command that uses an XLATE option, they must be correct for the specified direction. FUP does not try to adjust them.
- Translation occurs only if you use XLATE, XLATEIN, XLATEOUT, EBCDICIN, or EBCDICOUT options.
- An XLATE option overrides the EBCDICIN or EBCDICOUT options if you specify both on the same command.

CONFIG[URE] Examples

- To use CONFIGURE to set the default options for a DUP[LICATE] process:

```
-CONFIGURE DUP PURGE, SAVEID
-DUP A, B
FILES DUPLICATED:  1
```

  The first command (CONFIGURE DUP PURGE, SAVEID) declares the DUP[LICATE] configure options. The second command (DUP A, B) uses the PURGE and SAVEID options implicitly. Then FUP displays the number of files that are duplicated (1).

- To explicitly override a configured option:

```
-CONFIGURE DUP PURGE, SAVEID
-DUP A, B, SOURCEDATE
```

  The SOURCEDATE option in the DUP[LICATE] command overrides the configured SAVEID option.

- To display and reset CONFIGURE options:

```
-RESET CONFIGURE DUP
-SHOW CONFIGURE DUP
-CONFIGURE DUP PURGE, SAVEID
-SHOW CONFIGURE DUP
 DUP PURGE, SAVEID
```

  The first command (RESET CONFIGURE DUP) resets the CONFIGURE DUP[LICATE] options, and the second command (SHOW COFIGURE DUP) displays the options. (There are none.) The third command sets the new CONFIGURE options (CONFIGURE DUP PURGE, SAVEID), and the fourth command (SHOW CONFIGURE DUP) displays them (DUP PURGE, SAVEID).

- To use the CONFIGURE options from the previous example to demonstrate the ECHO CONFIGURE option:

```
-CONFIGURE ECHO CONFIGURE
-DUP A, B, SAVEALL
CONFIG:  PURGE, SAVEID
FILES DUPLICATED:  1
```

  The CONFIGURE options that FUP reads (PURGE and SAVEID) are displayed. In this example, the SAVEALL option overrides the SAVEID option.

- To use and reset CONFIGURE options that do not have named defaults:

```
-CONFIGURE COPY UNSTRUCTURED
-COPY PART1, $MOOSE.*.PART1
-COPY PART2, $MOOSE.*.PART2
-RESET CONFIGURE COPY UNSTRUCTURED
```

  The CONFIGURE option (COPY UNSTRUCTURED) is reset to avoid applying it to any subsequent COPY commands accidentally.

- To duplicate all the files in volume $BIG to volume $BACKUP, terminating the DUP[LICATE] process if 10 severe errors occur:

```
-CONFIGURE ALLOW 10 SEVERE ERRORS
-DUP $BIG.*.*, $BACKUP.*.*
```

- To display the contents of a file containing 8-bit characters (FILE1) and the contents of a file of 7-bit characters (FILE2):

```
-CONFIGURE DISPLAYBITS 8
-COPY FILE1
-CONFIGURE DISPLAYBITS 7
-COPY FILE2
```

- To change the number of columns to 80:

```
-CONFIGURE REPORTWIDTH 80
```

- To declare a translation table name (`my_encrypt`) that is contained in the MYCRYPT.ENCRYPT file:

```
-CONFIGURE XLATE my_encrypt TEXT IN mycrypt.encrypt
```

  The MYCRYPT.ENCRYPT file is expected to contain two translation tables (256 bytes each) in one record.

- To declare a translation table name (belgianswiss) that is contained in the standard CHARMAPS translation tables and use it to translate a file from $TAPE:

```
-CONFIGURE XLATE belgianswiss CHARMAP
-COPY $TAPE, FILEB, XLATEOUT belgianswiss
```

  You do not have to specify the name belgianswiss if it was the last (or only) XLATE table declared. However, it must be defined in $SYSTEM.SYSTEM.ZCHARMAP.

- To customize your options for your FUP RELOAD commands (NO DEALLOCATE, RATE 50 percent, SLACK 20 percent):

```
-CONFIGURE RELOAD NO DEALLOCATE, RATE 50, SLACK 20
```

Commands Related to CONFIG[URE]

COMMAND	Function	Page
RESET	Resets default file attributes or CONFIG[URE] options	2-155
SHOW	Displays default file attributes or CONFIG[URE] options	2-180

COPY: Copy Form

Makes a record-by-record copy from one file to another (the Copy Form of the COPY command) or displays the contents of a file (the Display Form). The COPY command functions apply to Enscribe files and SPOOLER files (code 129).

Note. For information on displaying file contents with the COPY command, see COPY: Display Form on page 2-54.

You need to understand when to use the COPY command, DUP[LICATE], and LOAD commands:

- Use the COPY command to change file attributes, to copy files to or from nondisk devices, or to display records in a file (optionally in different formats).
- Use the DUP[LICATE] command to create identical copies of disk files.
- Use the LOAD command to create a structured disk file from scratch. It is much faster than the COPY command.

Use the Copy Form of the COPY command to make a record-by-record copy from one file to another.

```
COPY [ in-filename ] [ , [ out-filename ]
     [ , copy-option ] ... ]


copy-option is:

   control-option
   in-option
   out-option
   display-option

control-option is:

      COUNT num-records
      FIRST { ordinal-record-num                }
            { KEY { record-spec | key-value  } }
            { key-specifier ALTKEY key-value    }
      [ NO ] TITLE
      UNSTR[UCTURED]
      UPSHIFT
```

in-option is:

```
BLOCKIN in-block-length
[ NO ] COMPACT
EBCDICIN
RECIN in-record-length
REELS num-reels
[ NO ] REWINDIN
SHARE
SKIPIN num-eofs
TRIM [ trim-character ]
[ NO ] UNLOADIN
VARIN
```

out-option is:

```
BLOCKOUT out-block-length
DENSITYOUT density
EBCDICOUT
FOLD
PAD [ pad-character ]
RECOUT out-record-length
[ NO ] REWINDOUT
SKIPOUT num-eofs
TAPEMODE mode
[ NO ] UNLOADOUT
VAROUT
XLATE [ translation-table-name ]
XLATEIN [ translation-table-name ]
XLATEOUT [ translation-table-name ]
```

display-option is:

```
O[CTAL]
D[ECIMAL]
H[EX]
BYTE[S]
A[SCII]
NO HEAD
[ NO ] TITLE
```

in-filename

is the name of the file that is the source of the copy. This file can be a disk file, a nondisk device, a process, or a tape DEFINE name. The file can also be any type of file that FUP handles, including structured files, EDIT files (up to 99,999,000 lines), or SPOOLER (file code 129) files. You cannot use wild-card characters in *in-filename* or specify *qualified-fileset* for it.

If you omit *in-filename*, the IN file enabled for FUP is used. For example, this situation occurs when you use FUP interactively, causing the home terminal to be the IN file.

Note. For information on relative files, see COPY: Copy Form Guidelines on page 2-50.

---

out-filename

is the name of the file that is the destination of the copy. This file can be a nondisk device, a process, an existing disk file, or a tape or spool DEFINE name. The file can also be any type of file that FUP handles, including structured files, EDIT files, or SPOOLER (file code 129) files. You cannot use wild-card characters in *out-filename* or specify a *qualified-fileset* for it. For EDIT files, the maximum number of output lines is 99,999,000.

If you omit *out-filename*, the OUT file enabled for FUP is used. For example, this situation occurs when you use FUP interactively, causing the home terminal to be the OUT file.

Although existing data in *out-filename* is never overwritten during a COPY operation, the placement of the records being copied depends on the file type specified for the destination file.

Note. For more information on the placement of the records, see COPY: Copy Form Guidelines on page 2-50.

---

control-option

controls the method used for copying.

COUNT *num-records*

is the number of records to copy.

- If you omit COUNT and do not use the FIRST option, FUP copies all records from the first record through the end-of-file (EOF).
- If you omit COUNT and use FIRST, FUP copies all records from the file indicated by FIRST through the EOF.

```
FIRST { ordinal-record-num                }
      { KEY { record-spec | key-value } }
      { key-specifier ALTKEY key-value  }
```

names the starting record of the input file for the copy. If you omit FIRST, the copy starts with the first record of the input file.

ordinal-record-num

is the number of records (from the beginning of the file) to be skipped. The first record in a file is record zero. If you specify this option for an unstructured disk file, the copy begins at:

ordinal-record-num * *in-record-length*

Note. The actual reading begins with the first record in the source file.

KEY { *record-spec* | *key-value* }

specifies the primary-key value for the starting record of a disk file. FUP begins reading the input file at the record you name with KEY.

Specify *record-spec* as an integer in the range 0 through 4,294,967,295.

- For unstructured files, give the starting relative byte address for *record-spec*.
- For relative files, give the starting record number for *record-spec*.
- For entry-sequenced files, give the record address for *record-spec*.

To indicate the approximate position of the starting record for key-sequenced files, use *key-value*. Specify *key-value* as *string* or:

```
"[" { string } [ , string ] ... "]"
    { 0:255  } [ , 0:255  ]
```

You can specify a list of strings with each string enclosed by quotation marks or integers representing byte values in the range 0 through 255. You must enclose the list of strings and integers (if specified) in square brackets.

Note. The brackets in *key-value* syntax are enclosed in quotation marks to indicate that they are part of the parameter—not to indicate that the parameter is optional. Do not include the quotation marks when you type the brackets.

For example, specify a key value as the ASCII string “T905”, followed by a word containing the integer value zero, and a word containing the integer value nine:

```
[ "T905", 0, 0, 0, 9 ]
```

key-specifier

is a one-character or two-character string (located inside quotation marks) specifying the alternate key to use for positioning purposes.

ALTKEY *key-value*

specifies the alternate key of the starting record for a key-sequenced disk file. FUP begins reading the input file at the specified record and obtains

subsequent rows in alternate key order. Specify *key-value* for key-sequenced files as described in KEY { record-spec | key-value } on page 2-38.

`[ NO ] TITLE`

directs FUP to write a title line that includes the name of the file, the time of the COPY process, and the last modification time of the file. The title line is the first line of the OUT file, followed by a blank line. These two lines become part of the OUT file and are included in the RECORDS TRANSFERRED count. The default is NO TITLE.

`UNSTR[UCTURED]`

causes an unstructured open on a file. You can use this option for any unstructured disk file that is not in EDIT file format (or any Enscribe file structure) and for any structured file where you want FUP to ignore its file structure. This option is for disk files only.

You must use this option on partitioned files to copy partitions individually.

`UPSHIFT`

converts lowercase alphabetic characters to uppercase.

in-option

controls the handling of the copied input file. The *in-option* parameter is not normally used with structured disk files.

`BLOCKIN` *in-block-length*

specifies the number of bytes in an input block. This number can be a value in the range 1 through 32,767 indicating the actual number of bytes requested in a single physical read operation.

Input records are blocked if *in-block-length* exceeds *in-record-length*. Records of the specified length are extracted from the input block until the number of bytes extracted equals *in-block-length* or until the last input record is read.

The read count for the records in a block (with the exception of the last record in the block) is equal to *in-record-length*. If *in-block-length* is not an exact multiple of *in-record-length*, the last record extracted from a full block is a short record with a read count equal to the number of bytes extracted.

If you specify BLOCKIN and *in-filename* is a tape DEFINE, do not specify the BLOCKLEN attribute of the DEFINE. Otherwise BLOCKLEN must match BLOCKIN *in-block-length*.

If the BLOCKLEN value is greater than 32,767 (the largest block size that FUP can accommodate), FUP issues this error message and then aborts:

```
ERROR - TAPE DEFINE BLOCKLEN VALUE TOO LARGE
```

If BLOCKLEN does not match BLOCKIN, this error message appears:

```
ERROR - TAPE DEFINE VALUE FOR BLOCKLEN
        CONFLICTS WITH BLOCKIN
```

The value of *in-block-length* is set to the BLOCKLEN DEFINE value if you omit the BLOCKIN option and *in-filename* is a tape DEFINE with these attributes:

- LABELS set to labeled tape processing
- USE set to IN (or not specified)
- BLOCKLEN specified

If these are not the attribute settings, FUP uses the *in-record-length* value for *in-block-length* and reads each input record in a separate physical operation. Record deblocking does not occur.

`[ NO ] COMPACT`

controls whether zero-length records are ignored when reading the file. This applies only to any IN relative files. The default is COMPACT (empty records are ignored), and records that follow an empty record in the OUT file are renumbered.

If you specify NO COMPACT, COPY transfers empty records from the IN relative files.

If you select the COMPACT option and the source is a relative file that contains empty records, this message appears:

source file : EMPTY RECORD FOUND AND NOT TRANSFERRED

This message indicates that the target file has fewer records than the source file. It is issued only once, when the first empty record is encountered.

`EBCDICIN`

treats the file specified for *in-filename* as though it contains EBCDIC characters and translates the characters to their ASCII equivalents.

All the characters converted between EBCDIC and ASCII retain their own image—with these exceptions:

EBCDIC	ASCII
Logical OR	Exclamation point

EBCDIC	ASCII
Cent sign	Left square bracket
Exclamation point	Right square bracket
Logical NOT sign	Circumflex

If you specify EBCDICIN and *in-filename* is a tape DEFINE, the EBCDIC attribute of the DEFINE must be ON or unspecified. FUP sets the EBCDIC attribute of the DEFINE to OFF for the life of the FUP command.

If you specify EBCDICIN, FUP translates the data. If you omit EBCDICIN (but a tape DEFINE is set to request translation), labeled tape processing performs the translation.

`RECIN` *in-record-length*

specifies the maximum number of bytes in an input record.

If you include the RECIN option (and *in-filename* is a tape DEFINE), the RECLEN attribute of the DEFINE must either be unspecified or match the RECIN *in-record-length*.

If you include the RECIN option, the actual number of bytes in each input record (the read count) depends on whether you also specify the TRIM option:

- If you do not specify the TRIM option, the read count is the actual number of bytes in the input record. Although the read count for unstructured files that are not in EDIT file format is exactly *in-record-length* bytes, the last record of the file might be less. The read count is the number of bytes actually read for all other files.
- If you specify the TRIM option, every trailing *trim-character* is deleted from the input record. The read count includes only the significant characters (those not trimmed).

If you omit the RECIN option, FUP determines the *in-record-length* value:

- If *in-filename* is a tape DEFINE that has RECLEN specified, USE set to IN (or not specified), and LABELS set for labeled tape processing, *in-record-length* is set to the RECLEN value.
- If you specify *in-block-length* using the FUP BLOCKIN option or the BLOCKLEN attribute of a tape DEFINE (and it is less than or equal to 4096), that value is used for *in-record-length*. If you specify an *in-block-length* greater than 4096, the value of *in-record-length* is 4096.
- If you do not specify *in-block-length* and the input file is an unstructured file that is not in EDIT file format (or is a process), FUP uses 132.

- If you do not specify *in-block-length* with the FUP BLOCKIN option or the BLOCKLEN attribute of a tape DEFINE (and the input file is a structured disk file or a nondisk device), FUP uses the record length specified when the file was created—or at system generation.

REELS *num-reels*

indicates the use of multiple reels and sets the number of reels that make up *in-filename* (for unlabeled magnetic tape only). You cannot specify this option if *in-filename* is a tape DEFINE with LABELS set for labeled tape processing. You must use the appropriate tape DEFINE attributes to process multi-reel labeled tapes.

Note. For D-series and G-series RVUs, see the *Guardian User's Guide* for more information. This parameter does not apply to H-series RVUs because reel-to-reel tapes are not supported.

Specify *num-reels* as an integer in the range 2 through 255. The tape is read until *num-reel* occurrences. FUP always requests *num-reels* tapes because it can recognize the end of a multiple reel (unlabeled) tape file only by using the *num-reels* specification. At the end of each reel (except the last one, depending on the REWINDIN and UNLOADIN parameters), the tape is rewound and unloaded, and you are prompted for the next reel.

The FUP process expects two consecutive EOF marks, and it cannot accommodate multireel unlabeled tapes that are written on IBM systems (or any other systems) that do not adhere to the two consecutive EOF mark format.

For FUP to distinguish between the end of an intermediate volume and the EOF, the value of *num-reels* must be correct.

If you include the REELS option in the COPY command and *in-filename* is a tape DEFINE, FUP displays this error message:

```
ERROR - REELS PARAMETER NOT ALLOWED
        FOR LABELED TAPES
```

If you omit REELS, *in-filename* data transfer terminates when FUP encounters a single EOF mark.

Labeled-tape handling for multiple reels follows IBM or ANSI standards, which do not have the limitations imposed by FUP.

[NO] REWINDIN

specifies that the tape is rewound (or not rewound) when the EOF is read from the tape (for magnetic tape only). If you specify NO REWINDIN, the tape remains positioned without rewinding. The default is REWINDIN. (The tape is rewound.) This option also applies to labeled tapes.

SHARE

opens *in-filename* with a shared exclusion mode (for disk files only). Using SHARE lets you copy a file even if it is currently opened by another process (unless it is open with exclusive exclusion mode). If you omit SHARE (and *in-filename* is a disk file), the file is opened with protected exclusion mode.

SKIPIN *num-eofs*

moves the tape past *num-eofs* end-of-file (EOF) marks before the data transfer begins (for magnetic tape only). You cannot specify this option if *in-filename* is a tape DEFINE with LABELS set for labeled tape processing. You must use the appropriate tape DEFINE attribute to skip files on a labeled tape.

Note. For more information, see the *Guardian User's Guide .*

If you include the SKIPIN option (and *in-filename* is a tape DEFINE), FUP displays this error message:

```
ERROR - SKIPIN PARAMETER NOT ALLOWED FOR
        LABELED TAPES
```

Specify *num-eofs* as an integer in the range -255 through 255.

- If you specify a positive value, the tape is forwarded past *num-eofs* EOF marks and positioned immediately after the last EOF mark passed.
- If you specify a negative value, the tape is rewound over *num-eofs* EOF marks (-1 multiplied by *num-eofs*) and then moved forward and positioned immediately ahead of the last EOF mark passed.
- If you specify a value of zero, the SKIPIN option is ignored.
- If you omit the SKIPIN option, the tape remains at its current position, and the data transfer begins with the next physical record on tape.

TRIM [*trim-character*]

deletes any trailing characters matching *trim-character*. Specify *trim-character* as either a single ASCII character in quotation marks:

"*c*"

Or specify it as an integer in the range 0 through 255 that specifies a byte value:

```
{ 0:255 }
```

Note. The *trim-character* default is an ASCII null character (binary 0).

`[ NO ] UNLOADIN`

specifies whether the tape is unloaded after it is rewound (for magnetic tape only). The default is UNLOADIN. (The tape is unloaded after it is rewound.) This option also applies to labeled tapes.

`VARIN`

reads variable-length, blocked records. These records can be produced using the COPY command option VAROUT on page 2-48.

Each variable-length, blocked record begins with a word that contains the length of the record, and the read count equals the value of that length indicator. You cannot use the TRIM option with VARIN, and this option works only with tapes written by FUP.

`out-option`

controls the handling of the output file. *`out-option`* is not normally used with structured disk files.

`BLOCKOUT` *`out-block-length`*

sets the number of bytes in an output block. Specify *`out-block-length`* as an integer in the range 1 through 32,767. This value is the maximum number of bytes to be written in a single physical operation. If a value greater than 32,767 is specified, FUP issues this error message and then aborts:

```
ERROR - TAPE DEFINE BLOCKLEN VALUE TOO LARGE
```

If you specify BLOCKOUT and *`out-filename`* is not a tape DEFINE:

- If the output-block length is greater than the output-record length, output-record blocking occurs. The block is filled with *`out-record-length`* records until it contains *`out-block-length`* bytes or the last output record is encountered.
- If *`out-block-length`* is not a multiple of *`out-record-length`*, the last record in a full block is truncated.
- If the write count for a record is less than *`out-record-length`*, the output record is padded in the output block with trailing nulls.
- The actual number of bytes written in a physical operation is *`out-block-length`* for all blocks (except the last). If the last block is not full, the actual number of bytes written is equal to the number of records in the last block multiplied by *`out-record-length`*.

If you specify the BLOCKOUT option and *`out-filename`* is a tape DEFINE with these attributes, you must not specify the BLOCKLEN attribute of the DEFINE, or the BLOCKLEN attribute must match *`out-block-length`*:

- LABELS set for labeled tape processing

- USE set to OUT (or not specified)

If you do not specify BLOCKLEN, it is set to *out-block-length* for the life of the FUP command.

If you omit the BLOCKOUT option, and *out-filename* is a tape DEFINE with these attributes, then FUP sets *out-block-length* to the BLOCKLEN value of the DEFINE:

- LABELS set to labeled tape processing
- USE set to OUT (or not specified)
- BLOCKLEN specified

If these are not the attribute settings, FUP uses the write-count value for *out-block-length* and writes each output record in a separate physical operation. Record blocking does not occur.

`DENSITYOUT` *density*

indicates the recording density for the output tape file. This option applies only to tape drives that support multiple densities.

Specify *density* as:

Density	Recording Density
GCR or 6250	6250 bpi (bits per inch)
PE or 1600	1600 bpi
NRZI or 800	800 bpi (not used on 5130)

If you specify DENSITYOUT, and *out-filename* is a tape DEFINE with LABELS set for labeled tape processing, the DENSITY attribute of the DEFINE must either be unspecified or must match *density*.

If the DENSITY attribute is unspecified, it is set to *density* for the life of the FUP command.

`EBCDICOUT`

specifies that output characters be translated to their EBCDIC equivalents. If you omit EBCDICOUT, FUP does not translate output.

All characters converted between ASCII and EBCDIC retain their own image—with these exceptions:

ASCII	EBCDIC
Exclamation point	Logical OR
Left square bracket	Cent sign
Right square bracket	Exclamation point
Circumflex	Logical NOT sign

If you specify EBCDICOUT and *out-filename* is a tape DEFINE, the tape DEFINE EBCDIC value must be ON (or unspecified), or this message appears:

```
ERROR - TAPE DEFINE VALUE FOR EBCDIC CONFLICTS
        WITH EBCDICOUT
```

If you include the EBCDICOUT option, FUP performs the translation. If you omit the EBCDICOUT option and a tape DEFINE is set so that translation is requested, the translation is performed by labeled tape processing.

Note. A warning occurs if you specify EBCDICOUT and an XLATE option (and FUP uses the XLATE option). For more information about the XLATE option, see CONFIG[URE] on page 2-26.

`FOLD`

divides input records that are longer than *out-record-length* into as many *out-record-length* records as needed to copy the entire input record.

If the last record written because of a FOLD is shorter than *out-record-length* (and you specify PAD), padding occurs. If you omit FOLD, truncation might occur.

`PAD [ pad-character ]`

specifies that records containing fewer than *out-record-length* bytes are padded with the *pad-character* (up to the specified record length). Specify *pad-character* as one of:

- A single ASCII character inside quotation marks:

  `"c"`

- An integer in the range 0 through 255 that specifies a byte value:

  `{ 0:255 }`

Note. The *pad-character* default is an ASCII null character (binary 0).

`RECOUT out-record-length`

sets the maximum number of bytes in an output record.

If you specify the RECOUT option, the actual number of bytes written for each output record (the write count) depends on whether you also specify the PAD option:

- If you do not specify PAD, the write count is either the read count or *out-record-length* (whichever is less).
- If you specify PAD, the write count is *out-record-length*. If the number of input bytes is less than *out-record-length*, the record is padded with trailing pad characters.

- If the number of input bytes exceeds *out-record-length*, the input record is truncated at *output-record-length* bytes (unless you specify FOLD).

If you specify the RECOUT option and *out-filename* is a tape DEFINE with these attributes, you must not specify the RECLEN attribute of the DEFINE, or RECLEN must match *out-record-length*:

- LABELS set for labeled tape processing
- USE set to OUT (or not specified)

If the RECLEN attribute is unspecified, it is set to *out-record-length* for the life of the FUP command.

If you omit the RECOUT option, FUP determines the *out-record-length* value:

- If *in-filename* is a tape DEFINE that has RECLEN specified, USE is set to OUT (or not specified), and LABELS is set for labeled tape processing, *out-record-length* is set to the RECLEN value.
- If you specify an *out-block-length* using the FUP BLOCKOUT option or the BLOCKLEN attribute of a tape DEFINE (and it is less than or equal to 4096), that value is used for *out-record-length*. If you specify an *out-block-length* greater than 4096, the value of *out-record-length* is 4096.
- If you do not specify *out-block-length* and *out-filename* is an unstructured disk file (or a process), *out-record-length* is 132.
- If you do not specify *out-block-length* with the FUP BLOCKOUT option or the BLOCKLEN attribute of a tape DEFINE (and *out-filename* is a structured disk file or a nondisk device), FUP uses the record length specified when the file was created—or at system generation.

`[ NO ] REWINDOUT`

specifies whether the tape is rewound when the COPY command finishes. If you specify NO REWINDOUT, the tape remains positioned without rewinding. The default is REWINDOUT. (The tape is rewound.) This option also applies to labeled tapes.

`SKIPOUT` *num-eofs*

moves the tape past *num-eofs* end-of-file (EOF) marks before the data transfer begins (for magnetic tape only). You cannot specify this option if *out-filename* is a tape DEFINE with LABELS set for labeled tape processing. You must use the appropriate tape DEFINE attribute to skip files on a labeled tape.

Note. For more information, see the *Guardian User's Guide* .

If you include the SKIPOUT option (and *out-filename* is a tape DEFINE), FUP displays this error message:

```
ERROR - SKIPOUT PARAMETER NOT ALLOWED FOR
        LABELED TAPES
```

Specify *num-eofs* as an integer in the range –255 through 255.

- If you specify a positive value, the tape is forwarded past *num-eofs* EOF marks and positioned immediately after the last EOF mark passed.
- If you specify a negative value, the tape is rewound over *num-eofs* EOF marks (-1 multiplied by *num-eofs*) and then moved forward and positioned immediately ahead of the last EOF mark passed.
- If you specify a value of zero, the SKIPOUT option is ignored.
- If you omit the SKIPOUT option, the tape remains at its current position, and the data transfer begins with the next physical record on tape.

TAPEMODE *mode*

indicates the mode in which the tape is written (for streaming drives only). The *mode* can be either STARTSTOP or STREAM. If you do not specify TAPEMODE, the default STARTSTOP mode is used for the tape operation.

STARTSTOP

selects the start-stop mode for the tape operation. STARTSTOP is the default.

STREAM

selects the streaming mode for the tape COPY operation.

You must also specify the largest block size possible to receive the maximum performance for streaming mode. For example, to use streaming mode, specify BLOCKOUT 28,000.

[NO] UNLOADOUT

specifies whether the tape is unloaded after rewinding (for magnetic tape only). The default is UNLOADOUT. (The tape is unloaded after it is rewound.) This option also applies to labeled tapes.

VAROUT

writes variable-length, blocked records.

Each variable-length record is preceded by a one-word record-length indicator. The record-length word contains the record length in bytes. Although records can contain an odd number of bytes, the record-length word is always aligned on a word boundary.

The write count and the record-length word of a record are equal—even if the record is truncated. Truncation occurs if the record is either longer than RECOUT or longer than BLOCKOUT subtracted by two. (The record-length word requires two extra bytes.)

If the next record with its record-length word does not fit in the current block, VAROUT terminates the current block and begins a new block because the blocks cannot be spanned.

VAROUT terminates a block by writing a record-length word of –1 (%177777) to indicate that no more valid records are in the block and then pads the remainder of the physical block. VAROUT cannot write the record-length word of –1 when the previous record ends on the block boundary or when *out-block-length* is odd and only one byte remains in the block.

Although null records (records with a length of zero) are supported, VAROUT does not allow the PAD and FOLD options.

This sample block has three records (MYSELF, COMPUTERS, and INC.) and uses a BLOCKOUT length of %30 to illustrate VAROUT:

```
%000006   <-  Record-length word for record 1
M     Y
S     E        Record 1
L     F
%000011   <-  Record-length word for record 2
C     O
M     P
U     T        Record 2
E     R
S
%177777   <-  Record-length word for end of block
p     p   <-  Padding
```

The third record (INC.) could not be written in the sample block because its record-length word requires six bytes beginning on a word boundary. The VAROUT option terminates the block (because only four bytes remain) and writes record three to the next block.

```
XLATE [ xlate-table-name ]
XLATEIN [ xlate-table-name ]
XLATEOUT [ xlate-table-name ]
```

specifies a translation table for translating records. You can specify XLATE, XLATEIN, or XLATEOUT and optionally includes a translation table name (*xlate-table-name*). If a translation table name is not provided, FUP uses the one defined in the most recent CONFIG[URE] XLATE command. An error occurs if *xlate-table-name* is omitted and no translation tables are defined or if the translation table name is not a defined translation table.

If XLATE or XLATEIN are specified, FUP uses the in direction of the translation table. If XLATEOUT is specified, FUP uses the out direction of the translation

table. Every character in the IN file is translated using the specified table before it is written to the OUT file.

Note. A warning occurs if you specify EBCDICOUT and an XLATE option (and FUP uses the XLATE option). For more information on the XLATE option, see CONFIG[URE] on page 2-26.

display-option

specifies the display format of the file for the Display Form of the COPY command.

Note. For descriptions of the display option and the Display Form of the COPY command, see COPY: Display Form on page 2-54.

COPY: Copy Form Guidelines

- COPY accepts a default file name for the source file. You can copy items to a file without having to specify the terminal name.
- You can copy up to 99,999,000 lines to an EDIT file (ABT or later SPRs).
- When output is copied to TAPE, FUP always turns buffering on if the drive accepts it.
- When tape copy operations are performed, two EOF marks are written at the end of the COPY operation to denote the end of the reel.
- If the destination of the copy is an unstructured file, a relative file, or an entry-sequenced file with existing data, new data is appended to the end of the file. Existing data is not overwritten.
- If the destination of the copy is a key-sequenced file, COPY places the records in the appropriate location instead of appending them to the end of the file.
- If *in-filename* is in the EDIT file format, each text line is treated as a logical record with a count read attribute.
- Unstructured files are different. Each physical read (except possibly the last read of a file) returns exactly *in-record-length* bytes.
- To copy to an EDIT file or spooler (code 129) file, you can use the COPY command.
- You cannot use the COPY command to copy a multireel set of tapes created by a NonStop COBOL application. The COBOL program marks the end of a multireel set of tapes differently from FUP, forcing you use a COBOL program to copy these types of tapes.

- A COBOL program can read multireel tape files created by FUP COPY.

  Note. For more information, see these manuals:

  - *COBOL85 for NonStop(TM) Himalaya Systems Manual* (D-series RVUs)
  - *COBOL85 for NonStop Systems Manual* (G-series RVUs)
  - *COBOL Manual for TNS/E Programs* (H-series RVUs)
  - *COBOL Manual for TNS and TNS/R Programs* (H-Series RVUs)

- FUP COPY uses sequential block buffering when accessing *in-filename*. All record locks are ignored.

  Note. For more information, see the *Enscribe Programmer's Guide*.

- When you enter a COPY command with no options, FUP opens *in-filename* with the read-only access, protected exclusion mode, unless *in-filename* is a terminal. If *in-filename* is a terminal, FUP opens it with shared exclusion mode.

  If you include the SHARE option in the command (and *in-filename* is a disk file), *in-filename* is opened with the shared exclusion mode.

- FUP opens *out-filename* with the exclusive exclusion mode unless *out-filename* is a terminal.

  A physical read of *in-filename* reads one logical record. A logical record is 132 bytes for unstructured files or files in the EDIT file format. A logical record is the length you specify for structured disk files and nondisk devices.

- A physical write to *out-filename* writes one logical record. The actual number of bytes written is the number of bytes read.

- Although you can specify a block size of as many as 32,767 bytes for the BLOCKIN and BLOCKOUT options, some peripheral devices have smaller maximum block sizes that must not be exceeded when you use the COPY command.

- If you copy a file that contains data records and zero-length records (empty records) to a relative output file, all the records are written, including the zero-length records, unless the input file is also a relative file.

  If the input file is a relative file, the zero-length records are skipped unless you specify NO COMPACT. To transfer zero-length records from a relative file, include NO COMPACT in the COPY command.

  For example, when you copy a relative file containing a combination of eight data records and two zero-length records, the record count of the output file is eight records instead of ten. So you lose zero-length records if you copy data with zero-length records in and out of a relative file.

- The [NO] COMPACT option affects copy operations for relative IN files only. If you specify NO COMPACT when you are copying another type of file, this message appears:

  ```
  WARNING - COMPACT OPTION IGNORED FOR NONRELATIVE FILES
  ```

- If the AUDIT option is set for *out-filename*, the copy request fails with file-system error 75 (requesting process has no current process transaction identifier).

- Be careful when using the PAD and TRIM options in a FUP COPY or LOAD operation. If it contains *trim-character* or *pad-character*, data can be altered or lost.

  If you pad each record in a data file with zeros to a standard size in bytes, store the records in another file, then trim the trailing zeros when you FUP COPY or LOAD the stored records. Any original data ending with zero is also trimmed.

  To avoid this problem, use a *pad-character* or *trim-character* that is not contained in your data.

- If you enter the DENSITYOUT parameter for a tape drive that is not a model 5106 or 5130, the COPY command continues without setting the tape density, and this message appears on the home terminal of the FUP process:

  ```
  WARNING -filename: DRIVE DOES NOT SUPPORT DENSITY
                     SELECTION
  ```

- If you enter the TAPEMODE parameter to a tape drive that does not support the setting, this warning message appears on your terminal, and the COPY operation continues without trying to set the tape mode:

  ```
  WARNING - filename: DRIVE DOES NOT SUPPORT TAPEMODE
                      SELECTION
  ```

- The COPY command cannot copy SQL files that are not SQL object files. You must use SQLCI COPY instead.

COPY: Copy Form Examples

These examples demonstrate using the COPY command without tape DEFINES.

Note. For examples of using the COPY command with tape DEFINEs, see the *Guardian User's Guide* .

- To copy the first 500 records of MYFILE to YOURFILE (two unstructured files in the current default subvolume), where input records are 80 bytes and the data is copied in 800-byte physical reads:

  ```
  -COPY MYFILE, YOURFILE, COUNT 500, RECIN 80, BLOCKIN 800
  ```

- To copy records from RELFILE (a relative IN file) to FILEB (FUP transfers zero-length records and records containing data):

  ```
  -COPY RELFILE, FILEB, NO COMPACT
  ```

- To output the contents of the EDIT file MYTEXT to the printer $LASER:

```
10> FUP / OUT $S.#LASER, NOWAIT/ COPY MYTEXT
```

- To copy a series of files onto magnetic tape, use a FUP COPY command for each file and include the NO REWINDOUT option in each command except the last one:

```
-COPY FILE1, $TAPE, NO REWINDOUT
-COPY FILE2, $TAPE, NO REWINDOUT
-COPY FILE3, $TAPE
```

- To return the files copied in the previous example from tape to disk:

```
-COPY $TAPE, NFILE1, NO REWINDIN
-COPY $TAPE, NFILE2, NO REWINDIN
-COPY $TAPE, NFILE3
```

- To copy two records from the key-sequenced FILEX to FILEZ with a primary key value specifying the starting record for FILEX:

```
-COPY FILEX, FILEZ, COUNT 2, FIRST KEY [ "U","P",89 ]
```

- To copy all the files that have changed since a specified date (01/01/01) to a backup tape:

```
-COPY $MOD.*.* WHERE MODTIME > 1 JAN 2001, $TAPE
```

- To read records from a file (PERSONAL), translate them using a specific translation table (MY_ENCRYPT), and copy them to another file (SECRET):

```
-COPY personal,secret,XLATE MY_ENCRYPT
```

- To give the file you are copying a title and display four lines of the file (including the title) on the terminal screen:

```
-COPY FILEZ, TITLE, COUNT 4
```

After you enter the command, the terminal displays:

```
$BASE.SAMPLE.FILEZ    12MAY01  DATA MODIF:  26APR2001  14:19

This is a sample file to show how the TITLE option within FUP
COPY works.
4 RECORDS TRANSFERRED
```

Commands Related to Copy: Copy Form

COMMAND	Function	Page
DUP[LICATE]	Creates identical copies of disk files	2-61
LOAD	Creates a structured file from scratch	2-129
CONFIG[URE]	Sets options for the COPY command	2-26

COPY: Display Form

Displays the contents of a file (the Display Form of the COPY command) or makes a record-by-record copy from one file to another (the Copy Form). The COPY command functions apply to Enscribe files and SPOOLER files (code 129). This subsection describes the Display Form.

Note. For information about making a record-by-record copy from one file to another with the COPY command, see COPY: Copy Form on page 2-35.

To display files containing 8-bit characters, use the DISPLAYBITS option (from the CONFIGURE command) with COPY.

Note. For more information about the DISPLAYBITS option and files with 8-bit characters, see CONFIG[URE] on page 2-26.

To display the contents of an Enscribe or SPOOLER file, use the Display Form of the COPY command. This subsection describes only *display-option* syntax. For a complete COPY syntax description, see COPY: Copy Form on page 2-35.

```
COPY [ in-filename ] [ , [ out-filename ]
     [ , display-option ] ... ]


display-option is:

   O[CTAL]
   D[ECIMAL]
   H[EX]
   BYTE
   A[SCII]
   NO HEAD
   [ NO ] TITLE
```

display-option

specifies the display format for the file. If you omit *display-option*, no formatting or conversion occurs, and each record is displayed as it appears in *in-filename*.

OCTAL or O

displays in octal and ASCII format.

DECIMAL or D

displays in decimal and ASCII format.

HEX or H

displays in hexadecimal and ASCII format.

BYTE

displays in byte and ASCII format. The two bytes of each word are converted separately.

- If you do not specify BYTE, a word is treated as a single value and converted accordingly.
- If you specify BYTE (but not OCTAL, DECIMAL, or HEX), the display is in byte-octal format.

ASCII or A

displays in ASCII format. This option is ignored when combined with any preceding *display-option*.

NO HEAD

omits the heading preceding each record displayed.

[NO] TITLE

directs FUP to write a title line that includes the name of the file, the time of the COPY process, and the last modification time of the file.

Copy: Display Form Listing Format

Example 2-1 shows the format FUP COPY uses to display files.

Example 2-1. COPY Command Listing Format

```
[ filename RECORD rec-num KEY xx ( %yy )
                LEN length-value mm/dd/yy hh:mm ]

offset word 0 word 1 ... word n  [ ASCII format ]
  .      .      .          .     [     .         ]
  .      .      .          .     [     .         ]
  .      .       .           .     [     .          ]
```

The headers and variables in Example 2-1 are:

filename	is the name of the file being displayed.
RECORD *rec-num*	indicates the ordinal number of the record that follows.
KEY *xx* (%*yy*)	(for disk files) indicates the primary key value (current record pointer) of the record in decimal (*xx*) and octal (%*yy*).
	(for unstructured files) *xx* is a relative byte address.
	(for relative files) *xx* is a record number.
	(for entry-sequenced files) *xx* is a record address.
	(for key-sequenced files) *xx* is not given.
LEN *length-value*	indicates the decimal length of the record in bytes.

mm/*dd*/*yy*	is the current date (displayed for the first record only).
hh:*mm*	is the current time (displayed for the first record only).
offset	is the offset from the beginning of the `word 0` record.
word 0...*word n*	is a block of *n* contiguous words of *filename*. If *out-record-length* is less than 100, *n* = 7. If *out-record-length* is in the range 100 through 120, *n* = 9; otherwise, *n* = 11.
ASCII-format	is the ASCII representation of the line to the left. Nonprintable characters are represented by a period (.). If you specify only ASCII (or A), the display is in ASCII format, and *offset* is given in decimal representation.

If you specify more than one number base (OCTAL, DECIMAL, or HEX), each line is displayed in the specified format and in ascending order of base values (O = 8, D = 10, and H = 16).

Copy: Display Form Examples

- To display the contents of a file (in both OCTAL and ASCII format) from the default volume and subvolume (MYFILE):

  ```
  -COPY MYFILE, OCTAL
  ```

- To open a file (MARYHAD), display six records starting with record 20 and display the first 20 characters in each record in octal format:

```
 -COPY MARYHAD, FIRST 20, COUNT 6, RECIN 20, OCTAL

$BASE.FUPDOC.MARYHAD  RECORD 20  KEY 2048  (%4000)  LEN 20
   0: 046541 071171 020150 060544  020141 020154 Mary had a l
  %6: 064564 072154 062440 066141                ittle la

$BASE.FUPDOC.MARYHAD  RECORD 21  KEY 2048  (%4000)  LEN 15
   0: 044564 020146 067554 066157  073545 062040 It followed
  %6: 064145 071000                              her

$BASE.FUPDOC.MARYHAD  RECORD 22  KEY 2048  (%4000)  LEN 17
   0: 052157 020163 061550 067557  066040 067556 To school on
  %6: 062440 062141 074400                       e day

$BASE.FUPDOC.MARYHAD  RECORD 23  KEY 2048  (%4000)  LEN 17
   0: 052157 020163 061550 067557  066040 067556 To school on
  %6: 062440 062141 074400                       e day

$BASE.FUPDOC.MARYHAD  RECORD 24  KEY 2048  (%4000)  LEN 15
   0: 044564 020146 067554 066157  073545 062040 It followed
  %6: 064145 071000                              her

$BASE.FUPDOC.MARYHAD  RECORD 25  KEY 2048  (%4000)  LEN 17
   0: 052157 020163 061550 067557  066040 067556 To school on
  %6: 062440 062141 074400                       e day
```

CREATE

Creates a disk file with the current file-creation attributes defined by the SET command. To override the current file-creation attributes defined by the SET command, include *create-param* in the CREATE command.

```
CREATE filename [ , create-param ] ...
```

filename

is the name of the file to be created. A partial file name is expanded using the current default names for system, volume, and subvolume.

create-param

overrides the current file-creation attribute setting that corresponds to this creation. The *create-param* used in a CREATE command does not change the file-creation attributes defined by the SET command.

Note. For descriptions of the *create-param* options, see SET on page 2-165.

CREATE Guidelines

- Extent size is rounded up at creation as necessary so it can hold an integral number of blocks.
- For files on disk drives in a disk drive enclosure, there are additional considerations. See EXT { extent-size } { (pri-extent-size , sec-extent-size) } on page 2-167.
- During the file-creation process, DP2 rounds up the extent size (to 2 pages or 4,096 bytes) because the extent size of DP2 files must always be an integral multiple of the BUFFERSIZE (for unstructured files) or of the BLOCK size (for structured files).

  To create an unstructured DP2 file with one-page extents, you must specify a BUFFERSIZE of 2048 bytes with either the FUP SET or FUP CREATE command.
- To take advantage of the REFRESH, DCOMPRESS, and ICOMPRESS options of the CREATE command, you must create primary and alternate-key files separately. The CREATE command passes these attributes to the primary file only when it is created.
- If you enter a CREATE, AUDIT command and the volume where the primary partition, secondary partitions, or alternate-key files reside is not audited, file-system error 80 (invalid operation on audited file or nonaudited disk volume) is returned. If you create an alternate-key file with the NO UPDATE option (and the key file is not on an audited volume), the error does not occur.

- If FUP terminates because an error occurs while CREATE is automatically creating alternate-key files, the primary-key file and any alternate-key files already created are not deleted. You must purge these files manually.
- CREATE cannot create SQL files that are not SQL object files. You must use SQLCI CREATE instead.
- To override the default SMF assignment on a CREATE command of a file on a logical volume, use the CREATE command with the PHYSVOL option.
- All partitions of a file are created with the same format version. For files that are not sequenced, an independently created partition must have the same format as all other partitions of the file.

CREATE Examples

- To create a non-partitioned unstructured file on a disk drive in a disk drive enclosure:

```
-SET TYPE U
-SET EXT (2,20)
-SET REC 80
-CREATE TEST
```

Note. DP will round the extent sizes up to multiples of 14 so the actual extent information will be ext (14,28), not ext (2,20) as specified.

- To create a partitioned unstructured file where one or more partitions reside on a disk drive in a disk drive enclosure.

```
-SET TYPE U
-SET EXT (14,42)
-SET REC 80
-SET PART (1,$FIBRE,14,42)
-CREATE TESTPART
```

Note. DP2 requires that the *pri-extent-size* and *sec-extent-size* of partitioned unstructured files explicitly be multiples of 14. It further requires that the pri-extent-size for every partition be the same; the *sec-extent-size* for every partition must also be the same for every partition. If the extent size is not the same for every partition, error 21 is returned.

- To create a key-sequenced file named MYFILE in the current default volume and subvolume with the file-creation attributes that are specified by the SET commands:

```
-SET TYPE K
-SET REC 80
-SET KEYLEN 8
-CREATE MYFILE
```

- If you include *create-param* in the CREATE command, the values you specify override the current file-creation attributes. (For more information, see SET on page 2-165.) To override the current attributes and create a relative file with a record length of 10, define an alternate key on the first five bytes of the record and have the alternate key reside in the alternate-key file SECFILE:

  ```
  -CREATE MYFILE, TYPE R, REC 10, ALTKEY ("AA", FILE 0, &
  -KEYLEN 5) , ALTFILE (0, SECFILE)
  ```

- To create a partitioned alternate key-sequenced file from the existing alternate key-sequenced file, add a partition to the existing file:

  1. Ensure the file is closed:

     ```
     -LISTOPENS FILE
     ```

  2. Set file-creation attributes to match those of the existing file:

     ```
     -SET LIKE $VOLnn.Subvol.first-alternate-keyfile
     ```

  3. Add the partitioning specifics:

     ```
     -SET PART (1,$VOLnn, primary EXT, secondary EXT, "PR
     altkeyvalue1"

     -SET PART (2,$VOLnn, primary EXT, secondary EXT, "PR
     altkeyvalue2"
     ```

  4. Purge the existing alternate key file:

     ```
     -PURGE $VOLnn.Subvol.first-alternate-keyfile
     ```

  5. Create the new partitioned alternate key file:

     ```
     -CREATE $VOLnn.Subvol.partitioned-alternate-keyfile
     ```

  6. Load the newly created partitioned alternate key file:

     ```
     -LOADALTFILE 0,$VOL.Subvol.filename, loadaltfile-command-
     options
     ```

  7. Restore file-creation attributes to their default settings:

     ```
     -RESET
     ```

  8. For each alternate key file you want to partition, repeat Steps 2 through 7.

- To create a partitioned alternate key file that is not key-sequenced from an existing alternate key file, use the ALTER command:

  ```
  -ALTER filename,PART ( sec-partition-num , [\ node.]$ volume[
  , pri-extent-size [ , sec-extent-size ] ] )
  ```

- To create a partitioned alternate key-sequenced file from scratch, use the SET:

  ```
  > FUP
  -set type k
  -set code 1001
  -set ext (32,8)
  ```

```
-set rec 54
-set block 4096
-set keylen 2
-set altkey ("LO",keyoff 42,keylen 4)
-set altkey ("VN",keyoff 46,keylen 8)
-set altfile (0,invalt)
-set part (1,$ade001,5,5)
-create inv
CREATED - $STORE1.SVOL1.INV
CREATED - $STORE1.SVOL1.INVALT
```

Commands Related to CREATE

COMMAND	Function	Page
ALTER	Modifies attributes of a file	2-9
RESET	Resets default file attributes or CONFIG[URE] options	2-155
SET	Sets default file attributes	2-165
SHOW	Displays default file attributes or CONFIG[URE] options	2-180

DEALLOCATE

Deallocates any file extents beyond the one that includes the end-of-file (EOF) address of the specified disk files. This command applies only to Enscribe files.

```
DEALLOCATE fileset-list [ , PARTONLY ]
```

fileset-list

is a list of files whose unused extents (beyond the EOF extent) are to be deallocated. If a file has the CLEARONPURGE option set, the data is physically deleted from the deallocated extents. You can specify *qualified-fileset* for this *fileset-list*.

PARTONLY

specifies the deallocation of any unused extents for all the primary or secondary partitions of partitioned files in *fileset-list*. If you reference a primary partition name, only the extents are deallocated from the primary partition. If you omit PARTONLY, extents are deallocated only for entire partitioned files, the primary partition must be specified in *fileset-list*, and it is an error to reference a secondary partition in *fileset-list*. PARTONLY has no effect on nonpartitioned files.

DEALLOCATE Guidelines

- DP2 does not let unused extents of audit-trail files be deallocated. Audit-trail files are identified by file code 134. Any attempt to deallocate unused extents of audit-

trail files are rejected with file-system error 2 (operation not allowed on this type of file). To let FUP continue after receiving such errors, use the FUP ALLOW ERRORS command.

- DEALLOCATE works only with SQL files that are SQL object files. On other SQL files, you must use SQLCI ALTER instead.
- If you specify the CLEARONPURGE option (with the SECURE command) for a file, a subsequent DEALLOCATE command physically clears the data from the deallocated extents of the file.
- To deallocate volume directory extents, you must use this file name syntax for *fileset-list*:

  $*volume*.SYS00.DIRECTRY

DEALLOCATE Example

To deallocate the unused extents past the EOF of MYFILE:

```
-DEALLOCATE MYFILE
```

Commands Related to DEALLOCATE

COMMAND	Function	Page
ALLOCATE	Allocates extents for a file	2-6
PURGEDATA	Purges the data from a file	2-144
REVOKE	Unsets the CLEARONPURGE file attribute	2-159
SECURE	Sets the CLEARONPURGE file attribute	2-161

DISPLAYBITS

Lets the COPY, INFO, and SHOW commands display 8-bit characters.

The DISPLAYBITS command became an option of the CONFIG[URE] command with the D30 product version of FUP. However, for compatibility purposes, any FUP product versions prior to the D30 RVU continue to recognize the DISPLAYBITS option as command syntax. For more information, see CONFIG[URE] on page 2-26.

DUP[LICATE]

Copies disk files. This command applies only to Enscribe files.

You need to understand when to use the COPY, DUP[LICATE], and LOAD commands:

COPY	To change file attributes or copy files to or from nondisk devices
DUP[LICATE]	To create identical copies of disk files
LOAD	To create a structured disk file from scratch (much faster than COPY)

```
DUP[LICATE] from-fileset-list , to-fileset
    [ , RESTARTABLE [ restart-filename ] ]
    [ , rename-option ] ...
    [ , EXT [ extent-size ]
    [       ( pri-extent-size , sec-extent-size ) ]
    [ , KEEP |, NEW |, OLD |, PURGE ]
    [ , PARTONLY ]
    [ , SAVEALL |, SAVEID |, SOURCEDATE ]
    [ , PHYSVOL [ physvol ] ]

rename-option is:

   [ ALTFILE ( key-file-number , filename )            ]
   [ PART ( sec-partition-number , [ \node.]$volume    ]
   [       [, pri-extent-size [, sec-extent-size] ] )  ]
```

from-fileset-list

is a file-set list specifying the files to duplicate. FUP duplicates files one at a time. You can specify *qualified-fileset* for this *from-fileset-list*, but any qualifiers must occur before the *to-fileset* specification.

- Partial file names are expanded using the current default node, volume, and subvolume where necessary.
- If you specify the RESTARTABLE option, you can specify only a single file in *from-fileset-list*.
- If you use the wild-card or *qualified-fileset* option to specify the name of a file in *from-fileset-list*, you must specify the file name part of *to-fileset* with the (*) wild-card character.
- To duplicate a Safeguard protected file, you must have read-access to the file.

Note. For more information, see DUP[LICATE] Guidelines for Safeguard Files on page 2-68.

---

to-fileset

is a file set specifying the destination of the duplication. You cannot specify *qualified-fileset* for *to-fileset*.

- If you use a wild-card character or *qualified-fileset* option to specify the name of a file in *from-fileset-list*, you must specify the file name part of *to-fileset* with the (*) wild-card character.
- If you specify the file name part of *to-fileset* as an asterisk (*), each output file is given the disk file name of its corresponding input file.
- If you specify the subvolume of *to-fileset* as an asterisk (*), each output file is given the subvolume name of its corresponding input file.

- The AUDIT options of the files in *to-fileset* are reset regardless of the state of that option for the files in *from-fileset-list*.

RESTARTABLE [*restart-filename*]

specifies that FUP create *restart-filename* (as an unstructured disk file with file code 855). FUP maintains information in this file that describes the progress of the DUP operation. If a failure occurs, the RESTART command can use this information when continuing the operation.

restart-filename

is the name of the unstructured (informational) disk file FUP creates for the RESTARTABLE DUP operation. The name must not be the same as any file name in the specified subvolume, the source file name, or the destination file name.

If you specify only the file name part for *restart-filename*, the name is expanded using the current default names for volume and subvolume.

The restart file is not created until after the destination file is successfully created. If *restart file-name* is the same as either a source file or a destination file or is the same as any other file name in the subvolume, FUP issues an error 10 (file already exists). However, if the restart file name conflicts with an existing file name and the source file is empty, FUP creates a destination file but does not create a restart file and does not issue an error 10.

The restart file is purged when a DUP operation is completed successfully. If a RESTARTABLE DUP[LICATE] process fails, use the FUP RESTART command to complete the DUP operation.

The restart file can never reside on an optical disk volume.

If you do not specify *restart-filename*, FUP creates a file named ZZRSTART on your current subvolume.

If multiple RESTARTABLE DUP[LICATE] processes are running concurrently, each RESTARTABLE DUP[LICATE] process must use a different *restart-filename*.

rename-option

renames the secondary-partition volume and alternate-key files when they are duplicated. FUP creates the destination file with new names for the secondary-partition volumes and alternate-key files.

If you do not specify *rename-option*, FUP creates alternate-key files and partitions with names specified by the primary partition. This option applies only when FUP creates the destination file. It does not apply if the OLD option is enabled.

ALTFILE (*key-file-number* , *filename*)

specifies a new name for an alternate-key file. The ALTFILE parameters are described in the SET command description for ALTFILE options. The name of the alternate-key file (identified by *key-file-number*) is replaced with the new *filename*. The specified *key-file-number* should already exist in the source file. If it does not exist, FUP issues a warning.

You must specify the number for *key-file-number* displayed by an INFO DETAIL command for the source file. The number specified for *key-file-number* in a SET or CREATE command is invalid if FUP changed it to a standard DP2 number.

PART (*sec-partition-number* , [*node*.]$*volume*
[, *pri-extent-size* [, *sec-extent*-size]])

specifies a new name for a secondary partition of a partitioned file.

Note. For a description of *sec-partition-number*, *pri-extent-size*, and *sec-extent-size*, see PART on page 2-171.

If you specify PART, the name of the partition (indicated by *sec-partition-number*) is replaced by the new partition name you specify. The indicated *sec-partition-number* should already exist in the source file. If the name does not exist, FUP issues a warning.

You must specify the number for *sec-partition-number* displayed by an INFO DETAIL command for the source file. The number specified for *sec-partition-number* in a SET or CREATE command file is invalid if FUP changed it to a standard DP2 number.

You cannot use PART to specify a new key separator for key-sequenced partitioned files.

EXT

specifies a change to the extent sizes in the duplicate file, including the primary (*pri-extent-size*) and secondary (*sec-extent-size*) extents. This lets you change the physical aspect of how the file is allocated on disk.

KEEP

does not duplicate any files that exist in both *to-fileset* and *from-fileset-list*. If a file specified in *to-fileset* already exists, the corresponding file in *from-fileset-list* is not duplicated. FUP lists the names of the files that are not duplicated. Checking is by name only. No file attributes are checked.

FUP uses only one of these options: KEEP, NEW, OLD, or PURGE. If more than one is specified, FUP uses the last one specified.

NEW

specifies that no file in *to-fileset* exists. For each file in *from-fileset-list*, FUP creates a new file with characteristics identical to the corresponding input file. The default mode is NEW.

FUP uses only one of these options: KEEP, NEW, OLD, or PURGE. If more than one is specified, FUP uses the last one specified.

OLD

specifies that each file in *to-fileset* must already exist. The characteristics of each file in *to-fileset* must match the characteristics of the corresponding file in *from-fileset-list* in these areas (if applicable): file type, record size, data-block size, data and index compression mode, and key length and key offset.

The extent size of the files in *to-fileset* might differ. The files you specify for *to-fileset* must be large enough to contain the *from-fileset* data. Existing data in *to-fileset* is overwritten.

FUP uses only one of these options: KEEP, NEW, OLD, or PURGE. If more than one is specified, FUP uses the last one specified.

PURGE

purges any existing file in *to-fileset* that has the same name as a file in the *from-fileset*. If a file currently exists with the same name as a destination file, FUP purges the current contents of the file. FUP creates a new file, with identical characteristics, for each file in the *from-fileset-list*.

FUP uses only one of these options: KEEP, NEW, OLD, or PURGE. If more than one is specified, FUP uses the last one specified.

PARTONLY

specifies that if *from-fileset-list* defines any primary or secondary partitions of a partitioned file, only those partitions (and any nonpartitioned files) are to be duplicated. Entire partitioned files are not duplicated.

Secondary partitions defined by *from-fileset-list* are not duplicated unless you specify PARTONLY.

If you omit PARTONLY and define a primary partition in *from-fileset-list*, FUP duplicates the entire partitioned file (all partitions). If PARTONLY is not specified, it is an error to specify a secondary partition in *from-fileset*.

PHYSVOL [*physvol*]

specifies the physical volume where a file should be created. This option overrides any SMF parameters after the CREATE command creates a file on the virtual disk. The specified *physvol* must be included in the storage pool associated with the SMF virtual disk process.

SAVEALL

transfers the owner ID, security, and last modified timestamp of the files in *from-fileset-list* to *to-fileset* unchanged. If you omit SAVEALL, SAVEID, and SOURCEDATE, FUP sets the owner ID and default security of the user who is executing the DUP command (the user ID that you used for the current session) and sets the timestamp to the time the DUP[LICATE] process completes.

The SAVEALL option is equivalent to the SAVEID and SOURCEDATE options.

FUP uses only one of these options: SAVEALL, SAVEID, or SOURCEDATE. If more than one is specified, FUP uses the last one specified.

Note. For more information about the SAVEALL option, see DUP[LICATE] General Guidelines on page 2-66.

SAVEID

transfers the owner ID and security of the files named in *from-fileset-list* to *to-fileset* with no changes. If you omit SAVEID and SAVEALL, FUP sets the owner ID and default security to that of the user executing the DUP[LICATE] command (the user ID that you used for the current session).

FUP uses only one of these options: SAVEALL, SAVEID, or SOURCEDATE. If more than one is specified, FUP uses the last one specified.

SOURCEDATE

transfers the last modified timestamp of the files in *from-fileset-list* to *to-fileset* unchanged. If you omit SOURCEDATE and SAVEALL, FUP sets the timestamp to the time the DUP[LICATE] process completes.

FUP uses only one of these options: SAVEALL, SAVEID, or SOURCEDATE. If more than one is specified, FUP uses the last one specified.

DUP[LICATE] General Guidelines

- The DUP command opens the file to be duplicated with read-only access and with protected exclusion mode.

  Note. For information about file access and exclusion modes, see the *Enscribe Programmer's Guide*.

- The FUP DUP[LICATE] command cannot duplicate SQL files except for SQL object files. You must use SQLCI DUP instead.
- The DUP[LICATE] command duplicates a corrupt file (in its corrupt state) and displays this warning message:

  filename: FILE IS CORRUPT

- Using the DUP[LICATE] command with the EXT option does not change MAXEXTENTS. Before you use the DUP[LICATE] command, issue an ALTER MAXEXTENTS command to the source file if necessary to make the new file fit.
- If you use the DUP command to duplicate a primary or secondary partition file with the OLD option, you must also specify the PARTONLY option. For example:

  ```
  -DUP from-partition,to-partition,OLD,PARTONLY
  ```

  If you do not specify PARTONLY with OLD for a partition, FUP displays this error message:

  ```
  ERROR - NOT ALLOWED FOR PARTITIONED FILES
  ```

- If you use the DUP command to duplicate a file that is audited by TMF, the new file is not an audited file. To restore audited status to a file, use an ALTER AUDIT command.
- If you use the *OLD* parameter, you can use the DUP[LICATE] command and change the extent sizes. Before you use the DUP[LICATE] command, you must create the *OLD* file with the new extent sizes.
- The SAVEID and SAVEALL options:
  - Preserve the licensed attribute (for more information, see LICENSE (Super ID) on page 2-115) only if the process accessor ID of the current FUP is the super ID, and the target file resides on the node where FUP is running. If the target file is remote, no warning message appears.
  - Preserve the PROGID attribute (for more information, see SECURE on page 2-161) if the process accessor ID of the current FUP is the super ID (or if you are the owner of the source file), and the target file resides on the node where FUP is running. If the target file is remote, no warning message is given.
  - Preserve the state of Trust flag (for more information, see TRUST on page 2-188) only if the process accessor ID of the current FUP is the super ID (255,255).
  - Transfer the CLEARONPURGE attribute (for more information, see SECURE on page 2-161) to a local or remote target file. The target file does not need to be on the node where FUP is running. To restore the CLEARONPURGE attribute, use a SECURE command.
- When DUP encounters a broken file, the broken flag is duplicated, and this warning message appears:

  ```
  filename:  FILE IS BROKEN
  ```

- When an SQL object file is duplicated, the expiration date (NOPURGEUNTIL) is not preserved. The expiration date is set by either the NOPURGEUNTIL option of the FUP ALTER command or by the SQLCI NOPURGEUNTIL attribute.

- You can use the DUP command on queue files. For example, to increase the size of the queue files:

  ```
  -DUP oldqueuefile, newqueuefile [EXT (pri-extent-size, sec-extent-size)] [SAVEALL]
  ```

  You cannot use DUP to increase MAXEXTENTS. To alter the MAXEXTENTS of a file, use the FUP ALTER command.

- The maximum number of files in *fileset-list* that can be duplicated at a time is 2,147,483,647.

DUP[LICATE] Guidelines for Safeguard Files

- To duplicate a Safeguard protected file, you must have read access to the input file (*from-fileset-list*) and create access to the destination volume and subvolume (*to-fileset*).
- Unless you have create access (using Safeguard) to it, the target file is not created successfully. If you run a DUP command with the PURGE option but do not have create access to the target file, the original file is purged, and the target file is not created.
- If you duplicate a Safeguard protected file, the new file is not Safeguard protected unless volume or subvolume protection exists on the target subvolume. To restore or set Safeguard protection, use the Safeguard command interpreter (SAFECOM).
- Whenever you duplicate a Safeguard protected file (file-level protection), this warning message appears:

  ```
  WARNING - filename: NO SAFEGUARD PROTECTION
  ```

- If you duplicate a Safeguard protected file with the SAVEID or SAVEALL option, the owner of the old file (and the corresponding security of the file) is retained.
- If you duplicate a Safeguard protected file without the SAVEID or SAVEALL option, you become the new owner of the file (and its security is your default).
- DUP command supports queue files. For example:

  ```
  -DUP oldqueuefile, newqueuefile [EXT (pri-extent-size, sec-extent-size)] [SAVEALL]
  ```

  MAXEXTENTS cannot be increased by DUP. Use FUP ALTER to alter the MAXEXTENTS of the file.

DUP[LICATE] Examples

- To duplicate all the files in the current default subvolume (and in the MYSVOL subvolume) and place copies of the files (using the same file names) in the NEWSVOL subvolume:

  ```
  -DUP (MYSVOL.*, *), NEWSVOL.*
  ```

- To let the DUP operation on the source file (FILE1) be restarted if the initial DUP operation does not complete successfully:

  ```
  -DUP FILE1, FILE2, RESTARTABLE
  ```

  If the initial DUP operation does not complete successfully, restart the previous command:

  ```
  -RESTART
  ```

- To duplicate a file in a different volume and rename the first alternate-key file (file 0) of the destination file $VOL2.RECD.ALTFILE:

  ```
  -DUP $VOL1.RECD.DATA, $VOL2.*.* , ALTFILE (0, $VOL2.RECD.&
  -ALTFILE)
  ```

- To duplicate a partitioned file in a different node and rename the secondary partition of the destination file \TWO.$VOL2.SUB1.PARTFILE:

  ```
  -DUP $VOL1.SUB1.PARTFILE, \TWO.$VOL1.*.*, &
  -PART (1, \TWO.$VOL2)
  ```

  When you duplicate a partitioned file (in a different node or volume), to duplicate a secondary partition from a volume that does not exist on the destination node, you must include the PART option. To duplicate the secondary partitions, use these commands (on separate command lines):

  ```
  -DUP $VOL3.SUB1.PARTFILE, \TWO.$VOL2.*.*, PARTONLY
  -ALTER \TWO.$VOL1.SUB1.PARTFILE, PART (1, \TWO.$VOL2)
  ```

  The DUP command duplicates only the secondary partition of PARTFILE (to $VOL2 on node \TWO) because the PARTONLY option is specified.

  ALTER changes the primary partition for the file on \TWO to point to the secondary partition that was duplicated using PARTONLY in the first command.

- To duplicate all files on the current subvolume that end in FILE:

  ```
  -DUP *FILE, NEWSVOL.*
  ```

- To duplicate all the EDIT files (that begin with an S) from one subvolume to another and purge any old copies (if they exist):

  ```
  -DUP S* WHERE FILECODE=101, NEWSVOL.*, PURGE
  ```

  Note. Qualifiers occur before the *to-fileset* specification.

- To duplicate the files that have changed (since a specified date) in all the subvolumes to a backup volume:

  ```
  -DUP $MILK.*.* WHERE MODTIME>1JAN2001, $BACKUP.*.*
  ```

  Note. Qualifiers occur before the *to-fileset* specification.

- To duplicate a file, but specify a change to its primary and secondary extent sizes:

  ```
  -DUP SPECIALK, SUGARPOP, EXT (8,4)
  ```

Commands Related to DUP[LICATE]

COMMAND	Function	Page
COPY	Creates a record-by-record copy of a file	2-35
LOAD	Creates a structured file from scratch	2-129
RESTART	Restarts an interrupted DUP[LICATE] operation	2-157
CONFIG[URE]	Sets default options for the DUP[LICATE] process	2-26

EXIT

Stops the current FUP process and returns to the command interpreter.

```
E[XIT]
```

EXIT Guidelines

- To run the EXIT command, you can enter `E` or `EXIT`.
- The FUP process terminates when FUP reads the end-of-file (EOF) mark of the input file that you specified in your command to run FUP. You do not have to end a FUP command file with an EXIT command because of the EOF mark.
- Simultaneously pressing the CTRL and Y keys at the terminal is the same as an EOF. If you press CTRL-Y at the FUP prompt, FUP terminates after it displays:

  `EOF!`

EXIT Example

To terminate FUP and return control of the terminal to the command interpreter:

```
-EXIT
10>
```

FC

Modifies and reexecutes a specific command.

```
FC [ -num | num | string | quoted ]
```

-num

displays a command previous to the current command. For example, to modify and execute the third command prior to the current command, specify -3.

num

is the number of a command line. For example, to modify and execute the second command of the current FUP session, specify the number 2.

string

is the first characters of a previous command. For example, to display the most recent DUP command that starts with a volume name, enter FC DUP $.

quoted

is a string enclosed in either single or double quotation marks. FUP searches every character in the command buffer—not just the first characters—until it finds the string. For example, to display the most recent command that referenced the system \KAUAI, enter FC “\KAUAI”.

To edit the command, use the space bar and the backspace key to position the cursor under the text that you want to change. Do not use the arrow keys to move the cursor.

The FC command accepts these three command-editing characters:

R *replacement-string*

replaces characters in the command line (beginning with the character above the R) with *replacement-string*. *replacement-string* is terminated by // or Return.

I *insertion-string*

inserts *insertion-string* into the command line in front of the character above the I. *insertion-string* is terminated by // or Return.

D

deletes the character above the D. Repeat to delete more characters.

FC Guidelines

- If you do not use parameters, FC displays the last command line entered.
- When you finish editing the line or have no changes to make, press RETURN to execute the modified command.
- Use the HISTORY command to obtain line numbers.
- You must begin your correction explicitly with one of the subcommands if the first character of the change is I, D, or R. Type the D or R under the character to be deleted or replaced. Type the I under the character that follows the insert position.
- If you enter a string without a preceding subcommand, R is assumed.
- The subcommand begins its operation at the character positioned directly above it. FC displays the modified line and prompts for another subcommand.

- Specify more than one subcommand per line by separating the subcommands with a double slash (//).
- To abort the FC command and not execute the modified command, press the BREAK key or the CTRL and Y keys—or enter a double slash (//) in columns 1 and 2 and immediately press RETURN.

Note. For more information about the FC command, see the *Guardian User's Guide* .

FC Examples

- To change the DETAIL option in the first command to the STATISTICS option:

```
-FC
-INFO MYFILE, DETAIL
.             ddSTAT
-INFO MYFILE, STAT
.
```

- Spaces typed after the I or R subcommand are part of the text to insert or replace. To make more than one change per line by ending the text string with two slashes (//) and spacing over:

```
-FC
-DUP FILE1, FILE2
.         i3//  r14
-DUP FILE13, FILE14
.
```

Commands Related to FC

COMMAND	Function	Page
HISTORY	Displays previous FUP commands	2-79
!	Reexecutes a previous FUP command without modification	2-4
?	Displays a previous FUP command	2-5

FILENAMES

Lists the names of files that match the specified wild-card option. The FILENAMES command is similar to the FILES command, but the FILES command lists all the files in the specified subvolumes.

```
FILENAMES [ / OUT listfile / ] [ fileset ]
```

OUT *listfile*

names an existing disk file or a device to receive the listing output from the FILENAMES command. You can use either a standard file name or a spool

DEFINE name as the OUT file name for a FILENAMES command. If *listfile* is an existing file, FUP appends the output to the file.

Note. For more information about *listfile*, see Specifying Files on page 1-8.

fileset

specifies the file set to be listed. The *fileset* parameter can use the wild-card option in both the subvolume name and the file name. If *fileset* is not specified, it defaults to all files in the subvolume. You can specify *qualified-fileset* for *fileset*.

FILENAMES Example

To display the names of all the files containing OLT in all the subvolumes of the current volumes that start with FUPKIR:

```
-FILENAMES FUPKIR*.*OLT*
$BASE.FUPKIR
    NOLT       NOLTE      OLTR       ROLT       XOLT       XOLT2
$BASE.FUPKIRK
    NOLT       NOLT2      OLTE       OLTR       ROLT
```

Commands Related to FILENAMES

COMMAND	Function	Page
FILES	Displays the names of all files in a subvolume	2-74
INFO	Displays information about the files	2-80

FILES

Displays all file names associated with one or more subvolumes. The FILES command is similar to the FILENAMES command, which displays subsets of files within a subvolume.

```
FILES [ / OUT listfile / ] [ subvolset ]

subvolset:

   [[[ \node.]$volume.] subvolume | * ]
```

`OUT listfile`

names an existing file or device to receive the output of the FILES command. You can use either a standard file name or a spool DEFINE name as the OUT *listfile* for the FILES command.

If you omit the OUT option, output is displayed on the terminal. If *listfile* is an existing file, FUP appends the output to that file.

Note. For more information about *listfile*, see Specifying Files on page 1-8.

subvolset

names a subvolume or set of subvolumes whose file names are to be listed. If you omit *subvolset*, FUP lists the files that reside in the current default volume and subvolume. You can use a wild-card character for *subvolume*, but you cannot specify *qualified-fileset* for *subvolset*. (To specify *qualified-fileset*, use the FILENAMES command.)

If you omit *node*, FUP lists the designated file names on the current default node.

If you omit *volume*, FUP lists the designated file names on the current default volume. If you include *volume*, you must also include *subvolume* or an asterisk (*).

If you use an asterisk (*) in place of *subvolume*, FUP lists all the file names on the designated volume.

On a system running the D-series, if you omit both *subvolume* and the asterisk (*), you must also omit *volume*. FUP lists the file names in the current default subvolume on the current default volume.

If you include both *volume* and *subvolume*, *subvolset* must be a contiguous string with $*volume* separated from the *subvolume* by a period (.).

FILES Guidelines

- If you request information for all the subvolumes on a volume, FUP displays the file names in each subvolume (by object name within the subvolume name).
- The FILES command applies to all types of Enscribe and SQL files. File names are displayed for unstructured and structured Enscribe files and for all types of SQL files (tables, indexes, views, catalog tables, the indexes on SQL catalog tables, and SQL object program files).

FILES Examples

- To list the names of the files in your current default subvolume:

  ```
  -FILES
  ```

- To list the names of the files in each subvolume of the current default volume:

  ```
  -FILES *
  ```

- To list the names of all the files in each subvolume on the volume $VOL2:

  ```
  -FILES $VOL2.*
  ```

- To list the names of all files in the subvolume (JIMMY) of the volume ($ACES) on the node (\TENNIS):

  ```
  -FILES \TENNIS.$ACES.JIMMY
  ```

- This example assumes that the current default node is \SYS1, the current default volume is $VOL1, and the current default subvolume is SUBVOLA. The default subvolume also contains an Enscribe file and an SQL table, view, and index.

  To list the names of all four files in the previous examples, use any of these commands:

  ```
  -FILES
  -FILES SUBVOLA
  -FILES $VOL1.SUBVOLA
  -FILES \SYS1.$VOL1.SUBVOLA
  ```

Commands Related to FILES

COMMAND	Function	Page
FILENAMES	Displays the names of files	2-72
INFO	Displays information about the files	2-80

GIVE

Changes the owner of a file. This command applies only to Enscribe files. Only the current owner of the file (or the super ID, (255,255)) can execute the GIVE command for a file.

```
GIVE fileset-list ,
{groupnum , usernum | groupname.username }


   [ , PARTONLY ]
```

fileset-list

is a list of files whose ownership is to be given to another user. You can use wild-card characters, and you can specify *qualified-fileset* for *fileset-list*.

groupnum , usernum

is the group and user numbers of the user who is to be given ownership of the files.

groupname.username

is the group and user names of the user who is to be given ownership of the files.

PARTONLY

specifies (for partitioned files) that only partitions included in *fileset-list* are to be given. If you omit PARTONLY, only entire partitioned files whose primary partitions reside in *fileset-list* are given. PARTONLY has no effect on nonpartitioned files.

GIVE Guidelines

- If you try to use the GIVE command to change the ownership of a file protected by Safeguard at the disk-file level and are not the super-ID (255,255), you receive file-system error 199 (disk file is Safeguard protected). Instead, you must use the Safeguard command interpreter (SAFECOM).
- If you try to change ownership of a Safeguard-protected file at the subvolume level but you do not have owner permission, you receive file-system error 48 (security violation).
- If you try to use the GIVE command to change the ownership of a file that is not Safeguard protected but do not have purge permission on the file or are not the super ID (255, 255), you receive file-system error 48 (security violation).
- You cannot use the GIVE command on a file that is currently open with exclusive exclusion mode.

- If a file is given to another user while the file is open for a different process, the access rights of the process that has it open are not affected.
- If you give a program file whose PROGID bit is set, the GIVE command clears that bit. To set the PROGID bit, use the FUP SECURE command. If the bit is set, the accessor ID is set to the ID of the program file when the program is run.

Note. For more information, see the *Guardian Programmer's Guide.*

- Files that you give to another user remain in their original subvolume. To move a copy of a file to another subvolume, use FUP DUP. After you give a file to another user, you might not be able to duplicate it because you are no longer the owner.
- GIVE can only give SQL files that are SQL object files. For other SQL files, you must use SQLCI ALTER, CREATE, and SQLCI SECURE instead.
- If you do not own all the files in a subvolume that you specify in *fileset-list*, first use the ALLOW option (within the CONFIGURE command), to set the number of allowable errors high enough to complete the GIVE operation.

GIVE Examples

- To give ownership of all files in the current default subvolume to the user with user ID 8,1:

  ```
  -GIVE *, 8,1
  ```

- To give the files PROG1, PROG2, and LIB in the subvolume $WORK.ORG to the user whose user ID is MANUALS.MARTIN:

  ```
  -GIVE ($WORK.ORG.PROG*, $WORK.ORG.LIB), MANUALS.MARTIN
  ```

HELP

Lists the syntax of the FUP commands.

```
HELP [ / OUT listfile / ] [ command | ALL [, SYNTAX ] | NEWS
]
```

`OUT listfile`

names a file to receive the output of the HELP command. If you omit this option, the output is sent to the OUT *listfile* that is enabled for the current FUP session—usually your home terminal. You can use either a standard file name or a spool DEFINE name as the OUT *listfile* for the HELP command. If *listfile* is an existing file, FUP appends the output to the file.

Note. For more information about *listfile*, see Specifying Files on page 1-8.

command

is the name of a FUP command whose syntax you want to see. If you abbreviate the command name, FUP displays help for the first command that matches the characters entered.

`ALL`

lists the names of all FUP commands. The default is ALL.

`ALL,SYNTAX`

lists the syntax for all the FUP commands.

`NEWS`

provides a one-line description for each significant new FUP feature in the last several RVUs. This informal information source does not describe syntax and does not necessarily include every new feature. The newest edition of this manual contains comprehensive details and syntax for new commands or features.

HELP Examples

- To display the names of all FUP commands, enter HELP ALL (or HELP):

  ```
  -HELP ALL
  ```

- To write the syntax for all the FUP commands to the file MYHELP:

  ```
  -HELP /OUT MYHELP/ ALL,SYNTAX
  ```

- To display the syntax of the FUP CREATE command:

```
-HELP CREATE
CREATE filename [ , create param ] ...
   create param                              -- see SET
```

- To display the syntax of the FUP PURGE command:

```
-HELP PUR
PURGE { [ ! ] fileset [, fileset ] ... [ ! ] }
      { [ ! ] ( fileset [, fileset ] ... )
          [, [NO] LISTALL ] [ ! ]             }
```

HISTORY

Displays your previous FUP commands.

```
HISTORY [ / OUT listfile / ] [ num ]
```

`OUT` *`listfile`*

names an existing disk file or a device to receive the listing output from the HISTORY command. You can use either a standard file name or a spool DEFINE name as the OUT file name for a HISTORY command. If *`listfile`* is an existing file, FUP appends the output to the file.

Note. For more information about *`listfile`*, see Specifying Files on page 1-8.

`num`

is the number of previous commands to display.

HISTORY Guidelines

- If you omit *`num`*, FUP displays the last 10 commands.
- If *`num`* is greater than the number of commands in the history buffer, FUP displays all the commands in the buffer.
- The HISTORY command display shows line numbers for each command. You can use line numbers in the FC, !, or ? commands. Line numbers are not displayed anywhere else in FUP.
- The HISTORY command buffer can hold from 50 through 200 commands, depending on the size of the commands. After the buffer becomes full, the oldest command is discarded, as necessary, for each new command. Discarded commands are not available from the HISTORY command, the ! command, or the ? command.

HISTORY Example

To display the last four commands entered:

```
-HISTORY 4
  8:INFO, DETAIL
  9:CREATE NEWFILE
 10:DUP OLDFILE, NEWFILE
 11:HISTORY 4
```

Commands Related to HISTORY

COMMAND	Function	Page
FC	Modifies a previous FUP command	2-70
!	Reexecutes a previous FUP command	2-4
?	Displays a previous FUP command	2-5

INFO

Displays disk file characteristics of Enscribe files; SQL/MP and SQL/MX tables, indexes, or views; direct and SMF virtual disk files; and OSS files.

FUP INFO supports SQL/MX ANSI names. FUP converts each ANSI name to the corresponding list of Guardian file names and then performs INFO on each of these files.

Note. FUP support for fully qualified ANSI names for the INFO command is applicable on H06.04 and subsequent RVUs.

FUP short INFO for a TABLE or INDEX displays the information about all partitions of that particular table or index.

FUP short INFO for a TABLE PARTITION or INDEX PARTITION displays the information about the partition of the table or the index.

FUP long INFO for a TABLE or INDEX displays the information about a single partition as all the partitions contain the same information in the long form.

```
INFO [ / OUT listfile / ] [ fileset-list / ansiname-list ]
     [ , DETAIL                                          ]
     [ , EXTENTS                                         ]
     [ , STAT[ISTICS] [ , PARTONLY | PARTIAL num ] ]
     [ , USER { groupnum , usernum }
              { groupname.username }                     ]
```

OUT *listfile*

names an existing disk file or device to receive the listing output of the FUP INFO command. The OUT *listfile* defaults to your home terminal. You can use either

a standard file name or a spool DEFINE name as the OUT *listfile*. If *listfile* is an existing file, FUP appends output to that file.

Note. For more information, see Specifying Files on page 1-8.

fileset-list

is a list of disk files for which the file characteristics are displayed. The *fileset-list* can include Enscribe files, OSS files, and all types of SQL/MP and SQL/MX files (tables, indexes, views, catalog tables, and indexes on SQL/MP and SQL/MX catalog tables). If you omit *fileset-list*, the INFO command displays characteristics for all files in the current subvolume. You can specify *qualified-fileset* for *fileset-list*.

To display information for temporary files only, you must explicitly specify # in the file name. The file name can contain a wild card, as in `INFO #*`. You can use the asterisk (*) wild-card character alone or with other characters. For example, to display information for all temporary files that end in 1:

```
INFO #*1
```

The command INFO *.* does not display temporary files.

ansiname-list

ansiname-list = ‘ansiname’ [, ‘ansiname’]...

identifies SQL/MX ANSI name tables, indexes, partitions of tables and indexes, and any combination of these objects. A single quote (') is required to precede and delimit each ansiname. The ANSI names syntax is in accordance with Unified Syntax Proposal. The syntax is:

```
ansiname ::= {TABLE | INDEX}  base-mx-object-names

base-mx-object-names ::=  base-mx-object-name            |
             (base-mx-object-name [, base-mx-object-name …] )

base-mx-object-name ::= SQL-name [ partitions ]

partitions ::= PARTITION ( SQL-identifier [, SQL-identifier
…])
```

SQL-name

is used to name base SQL objects (such as tables or indexes) in addition to their SQL containers: catalogs and schemas. The names (called 3-part names) for SQL base objects such as tables, indexes, or modules are composed of three SQL identifiers separated by two dot characters (for example, CAT.SCH.T).

SQL-identifier

is a name used by SQL/MX to identify tables, views, columns, and other SQL entities. SQL identifiers can be either regular or delimited and can contain up to 258 characters in external form, or equivalently up to 128 characters in internal format. Regular identifiers begin with a letter (A through Z or a through z), but can also contain digits (0 through 9), or underscore characters (_).

Regular identifiers used to name a SQL/MX module (the basic object part) can start with the ^ character or contain the ^ character.

Note. The information regarding SQL/MX module provided above is for reference purpose only. FUP commands do not support the MODULE keyword.

A delimited identifier is enclosed in double quotes ("). Delimited identifiers are character strings that appear within double quote characters (") and consist of alphanumeric characters and other characters, except for character @, /,\, and ^. To include a double quote character in a delimited identifier, use two consecutive double quotes. A delimited module name in SQL/MX can contain the circumflex character (^).

DETAIL

gives detailed information on file characteristics (including SMF information). Use the DISPLAYBITS option (from the CONFIGURE command) with INFO,DETAIL when the file contains alternate keys containing 8-bit characters.

Note. For more information about the DISPLAYBITS option, see CONFIG[URE] on page 2-26.

EXTENTS

provides a listing of extent allocation by file (except SQL/MP and SQL/MX views). If specified, views are skipped.

STAT[ISTICS]

provides all the DETAIL information and statistical data on blocks and records for Enscribe-structured files, SQL/MP and SQL/MX tables, and indexes. Statistics information does not appear for unstructured Enscribe files, SQL/MP and SQL/MX program files, views, or shadow labels.

Note. The DETAIL information is not provided if the CONFIG STATONLY option is specified. For more information, see CONFIG[URE] on page 2-26.

PARTONLY

limits the information to any partitions that you specify explicitly in *fileset-list*. For example, if you specify only a secondary partition of an SQL table, statistical information about the primary partition (or any other secondary partition) does not appear.

PARTONLY is implied (for Enscribe files) if you specify a secondary partition because secondary partitions do not contain information about other partitions.

If you omit PARTONLY and specify the primary partition of an Enscribe file (or specify any partition of an SQL/MP and SQL/MX object), the STAT[ISTICS] option provides information about all partitions of the file.

PARTIAL *num*

specifies the percentage of the file used (read) to generate the statistics. You specify the percentage (*num*) of the file (from 0 through 100) that FUP reads and analyzes.

If PARTIAL is not specified (or if *num* is 0 or 100), FUP reads all the file. Because FUP reads the file in increments of 56 KB, the actual percentage used might be higher than the percentage requested.

USER

restricts the display to files in *fileset-list* owned by the user identified by *groupnum*,*usernum* or by *groupname*.*username*. If you include USER but omit *groupnum*,*usernum* and *groupname*.*username*, the display is restricted to the files in *fileset-list* that you own.

Note. Super-group users (255, n) do not have automatic access to OSS files.

INFO Guidelines

- If a transaction is still open, INFO specifies files that were opened and closed during the transaction as open. The files still have outstanding locks against them. The LISTOPENS command does not specify these files as open because the files do not have any openers.
- When a listed file is purged during the execution of the INFO command, INFO displays Error 11 (File not in directory).
- To perform a FUP INFO,STAT on volume directories, use this list-file syntax:

  $*volume*.SYS00.DIRECTRY

  For example, the command FUP INFO $DATA01.SYS00.DIRECTRY,STAT returns statistics on the volume $DATA01.
- FUP cannot show Safeguard protection for files protected at the volume or subvolume level. Additional discrepancies between Safeguard protection and what FUP displays are evident when files are copied to or from systems where the user has default Safeguard protection on only one system.
- Although the Safeguard product might not be currently running, FUP INFO always displays **** for the security vector of files individually protected by Safeguard.
- If you use INFO with the STAT option and FUP detects errors while generating the statistics, you receive an error (ERR 59). The error can occur if the file is being

updated concurrently, causing a transient structure problem that FUP encounters because it is reading large blocks.

- INFO with the STAT option does not detect each instance of a corrupted file.
- INFO with the STAT,PARTIAL option provides information without reading the entire file. The accuracy of STAT,PARTIAL depends on the condition of the file and how much of the file was analyzed. For example, statistics on index blocks are almost always unreliable when you use the PARTIAL option.
- FUP recognizes OSS files with the INFO command. This command is the only FUP command that handles OSS files.
- FUP INFO shows SMF information about files only in the DETAIL option. If the file is on a virtual disk, FUP displays the physical volume name of the file in a DETAIL display. If you omit the DETAIL option, FUP displays the standard information for SMF files on virtual disks.
- FUP does not display SMF files by physical file name unless the file name explicitly includes one of the reserved SMF subvolume names (`$physvol.ZYS*.*` or `$physvol.ZYT*.*`). In such cases, FUP displays the logical file name in the DETAIL display. If the DETAIL option is omitted, FUP displays its standard output for files in `ZYS*.*` and `ZYT*.*` subvolumes.
- FUP might append G (in the INFO listing) or GMT (in the INFO DETAIL listing) to the last modification date and time for Enscribe files or to any timestamp for SQL/MP and SQL/MX files. These letters indicate the displayed time is in Greenwich mean time (GMT) because the timestamp conversion failed to get the local time. If this occurs, an entry is probably missing from the system DST table. No indication means the time is displayed in local civil time.
- The user running FUP must have remote access to any system, which is implicitly referenced by the ANSI name used in the FUP command. For example, if the ANSI name is 'TABLE C.S.T', the user must have access to any node on which partitions of table C.S.T reside, and so on.
- The above explanation can be used to explain both an error 8551 from ANSI names or error 48 from the file system, depending on the command that was used.
- INFO supports SQL/MX objects, TABLE, INDEX, and PARTITIONS.

INFO Listing Format

Example 2-2 shows the format that the FUP INFO command (with no options) uses to display file information.

Example 2-2. INFO Listing Format

```
              CODE  EOF LAST MODIF OWNER RWEP TYPE REC BL
[\node.] $volume.subvolume
name  open-  code  eof  mod       owner sec  type rec bl
      state
```

name

is the disk file name of the file whose characteristics are being displayed.

A question mark (?) after the file name might appear for an OSS file entry for which FUP cannot read the last modification time.

open-state

is the open state of the file. It is displayed as any of:

null The file is not open, failed, or broken.

C The file is corrupt. A corrupt file is a file whose contents are in question. DUP and LOAD mark the destination file as corrupt while these operations are performed. If the operation does not complete normally, the file is marked corrupt and should be purged.

O The file is open, or a TMF transaction is active on the file.

? The file is crash-open. That is, it was open when a total system failure occurred or when the volume where it resides became unavailable.

R The file cannot be opened. Media recovery is needed (undo, redo, or rollforward).

B The file is open but received an I/O or consistency check failure and needs media recovery.

The states can occur in many combinations. For example:

OB The file is open but has an I/O or consistency check failure and needs a media recovery at some point.

?B The file is crash-open and broken.

code

is the file code. Software development has reserved file codes 100 through 999 for its own use. For a list of the file codes that are currently reserved, see Table 2-2, System File Code Definitions, on page 2-87.

CODE 0 (zero) is the default code for user-created files. It appears as a blank in the CODE column of the FUP INFO listing.

OSS designates OSS files.

Letters and symbols that appear after the code indicate:

A TMF audits the file.

L The file is licensed. For more information, see LICENSE (Super ID) on page 2-115.

P The PROGID attribute of the file is on. For more information, see SECURE on page 2-161.

+ The file is a Format 2 file.

eof

is the number of bytes contained in the file.

mod

is the date that the file was last written. If the file was modified today, the date is blank, and only the time of day is given. Otherwise, the year, month, day, and time are given. The year field is displayed with four digits.

The word QUESTIONABLE might appear for an OSS file entry for which FUP cannot read the last modification time.

owner

is the identification number of the file owner:

group-num , user-num

The super ID (255,255) is given as -1.

sec

is the security level assigned to the file (*rwep*):

r	Read
w	Write
e	Execute
p	Purge

Values for *rwep* are:

****	Safeguard protected (file mode only)
-	Local super ID only
O	Owner only (local)
G	Member of owner's group (local)
A	Any user (local)
U	Member of owner's user class—owner only (local or remote)
C	Member of owner's community—member of owner's group (local or remote)
N	Any user (local or remote)
*SQL	SQL/MX object

Note. For an OSS file, a 10-character OSS security vector appears in the RWEP column. (The vector extends into the TYPE column.) For more information, see the *Open System Services User's Guide*.

Note. If an OSS file has a POSIX ACL protection, FUP INFO displays a plus sign (+) after the permissions. However, if FUP INFO is executed remotely from a system without ACL support, "+" will not be printed for files with optional ACL entries. This feature is supported only on systems running G06.29 and later G-series RVUs and H06.08 and later H-series RVUs.

type

is one or more of:

null	Unstructured
R	Relative file structure
E	Entry-sequenced file structure
K	Key-sequenced file structure
*t*A	File has alternate key
P*t*	File is partitioned
XP*t*	File is an extra partition
Ta	SQL/MP or SQL/MX table
In	SQL/MP or SQL/MX index
PVi	SQL/MP or SQL/MX PView
SVi	SQL/MP or SQL/MX SView
Pg	A code 100 file with SQL/MP or SQL/MX compiled objects

where *t* can be one or more of R, E, K, P, or X.

rec

is the logical record length of the Enscribe file in bytes. For unstructured files, this field is blank.

bl

is the block length of the Enscribe file in kilobytes. For unstructured files, this field is blank.

Software development has reserved file codes, 47 through 22222 for its own use. Table 2-2 lists these codes and their corresponding definitions.

Table 2-2. System File Code Definitions (page 1 of 10)

File Code	Definition
OSS	OSS file
47	TMDS CLIP code dump file
68	MIS Batch file
69	MIS Batch file

Table 2-2. System File Code Definitions (page 2 of 10)

File Code	Definition
92	not yet determined
94	PRS employee file
96	PRS systems file
98	PRS product file
99	TTSI control file
100	TNS object file or OSS data file
101	EDIT-format file
102	TTEXT file
103	trace data file
105	TAL GLOBAL file
106	TAL Error log
107	Runtime data unit file
110	EDIT VS recovery file
111	EDIT VS stack dump file (data area image)
115	TEDIT TEDPROFL file
120-126	Spooler control files
127	Spooler data file
128	TVIEWER mark
129	Spooler job file
130	Inspect save file
131	INSPECT file
133	TMF control file
134	TMF audit-trail file
141	Compressed dump file
142	CPU dump file
143	143 SIERRA cpu dump file
144	Processor dump file (up to and including G06.15)
145	Processor dump file using RCVDUMP (as of G06.16)
146	Processor dump file using TFDS (as of G06.16)
150	QDDL/QRW RECDESC file
160	QDDL/QRW FILEDESC file
161-169	Workload Measurement System files
170	XRAYSCAN structured output files for Enform reports
175	Measure data file

Table 2-2. System File Code Definitions (page 3 of 10)

File Code	Definition
176	SQL/MP table file for Surveyor
178-179	ONGUARD control files
180	C data file
199	TACL cprules file
200	DDL (dictionary definition) file
201	DDL (alternate key) file
202	DDL (object definition) file
203	DL (object text) file
204	DDL (object build list) file
205	DDL (record definition) file
206	DDL (key definition) file
207	DDL (constant definition) file
208	DDL (object usage) file
209	DDL (token) file
210	Pathmaker INSTALLS file
223	Enable log file
230-232	ADA data file
249	TRANSFER (remote open control) file
250	Transfer profile file
251	Transfer session file
252	Transfer item descriptor file
253	Transfer recipient file
254	Transfer folder file
255	Transfer item data file
256	Transfer distribution list file
257	Transfer ready file
258	Transfer time file
259	Transfer network file
260	Transfer inverted folder file
261	Transfer restart file
262	Transfer name file
263	Transfer DIN file
264	Transfer alias file
265	Transfer trace file

Table 2-2. System File Code Definitions (page 4 of 10)

File Code	Definition
266	Transfer queue file
267	Transfer inverted attachment file
268	Transfer external objects file
269	TRANSFER (interest group) file
275	Transfer WORDLINK and Translator format name file
276	Transfer WORDLINK and Translator character map file
277	Transfer WORDLINK and Translator batch gateway configuration file
278	Transfer WORDLINK and Translator format type file
280	Transfer WORDLINK and Translator text server text file
281	Transfer System Management Monitor database monitor sample file
282	Transfer System Management Monitor queue monitor sample file
283-290	Transfer file
291	TRANSFER (P1-message id) file
292	TRANSFER (P2-message id) file
293	TRANSFER (P2 ITEMID) file
294	TRANSFER (alternate name file) file
295	TRANSFER (remote name file) file
296	TRANSFER (directory services data) file
297	TRANSFER (R Justify file) file
298	VIEWPOINT help text file or a TRANSFER (Depot Statistics) file
299	MHS Gateway Accounting file
300	TPS (Pathway) TCL program directory file
301	TPS (Pathway) TCL program code file
302	TPS (Pathway) SCREEN COBOL symbol file
303-304	TPS (Pathway) file
305	TPS (Pathway) TCP data area swap file
306	TPS (Pathway) AM control file
307	TPS (Pathway) Path TCP dump file
308	TPS (Pathway) Pathway trace file
309	TPS (Pathway) TCL program directory file
310-399	TPS (Pathway) PATHMON stack dump file
400	Tape simulator control file
401	Tape simulator data file
402	Consolidated Collected Performance Data file

Table 2-2. System File Code Definitions (page 5 of 10)

File Code	Definition
403	Psuedo Measure Structured file
404	Performance Reporting Data file
405	Kernel-Managed Swap Facility file
406	Tape IOP trace dump file
410	EXERCISE message file
411	EXERCISE error information file
412-419	EXERCISE files
420	INFOSAT/DATABOLT diagnostic log file
425	TMDS process database file
430	TMDS comms library file
430	EXERCISE Tandump segmented save file
431	TMF online dump file
432	BACKUP Dump files
440	TACL saved variable segment file
444	OSS file system mapping files
448	FATTY (Memory Tool) file
450	C00 file server and ViewPoint status display configuration file
451	Event display configuration file
460-462	SMS catalog files
470	Wire wrap list output file
480	SRL Registry Catalogue
481	SRL entry vector & initial instance data (SRLINIT)
482	SRL SRL-set load-set file (SRLSETL)
500	NonStop II processor microcode file
502	NonStop II microcode for SHADOW
505	5106 Tri-Density tape drive microcode object file
506	Microcode compiler symbols table
510	Standard (unformatted) microcode file
520	NonStop TXP processor microcode file
521	GASM-format microcode object file
522	Cyclone microcode file
523	Cyclone microcode file
524	Himalaya IOS microcode file
525	NonStop VLX processor microcode file

Table 2-2. System File Code Definitions (page 6 of 10)

File Code	Definition
530-536	NFS Configuration files
537-539	links
540	SAFEGUARD logical user file
541	SAFEGUARD user audit file
542	SAFEGUARD security database
543	SAFEGUARD object audit file
544	SAFEGUARD logging file
545-546	SAFEGUARD configuration file
547	SAFEGUARD pattern database
549	Encrypted file
550-565	SQL/MX files
566-568	SQL/MP file
569	SQL scratch file
571	NonStop SQL (catalog) file
572	NonStop SQL (base tables) file
573	NonStop SQL (columns) file
574	NonStop SQL (comments) file
575	NonStop SQL (constraints) file
576	NonStop SQL (files) file
577	NonStop SQL (indexes) file
578	NonStop SQL (keys) file
579	NonStop SQL (partions) file
580	NonStop SQL (programs) file
581	NonStop SQL (tables) file
582	NonStop SQL (transaction ids) file
583	NonStop SQL (dependancies) file
584	NonStop SQL (table version) file
585	NonStop SQL (views) file
586	NonStop SQL (cprules) file
587	NonStop SQL (cprlsrce) file
600	MUMPS global file
601	MUMPS routine file
602	MUMPS global directory file
603-620	MUMPS files

Table 2-2. System File Code Definitions (page 7 of 10)

File Code	Definition
621	(NS-SOG) Service Object file
622	(NS-SOG) Service Object Element file
623	(NS-SOG) Application file
624	(NS-SOG) Application Service Object file
625	(NS-SOG) Run-time Server Object file
626-629	(NS-SOG) Run-time Server Object file.
650	ENVISION (savework) file
651	ENVISION (hard db) file
652	ENVISION (map db) file
653	ENVISION (tmf db) file
654	ENVISION (model db) file
660	Encore capture file
661-664	Encore file
665	NETTACL TRACE file
666	NETTACL MOVIE file
667	ESCORT SYSDB
668	ESCORT MAPDB
669	Encore file
675	ES-FASTCONNECT log file
700	TNS/R native object file
701	PTAL GLOBAL file
703	pTAL file
706	pTAL file
710	CSS load module
711	PCSLAM manuals data file
720-723	RDF file
800	TNS/E native object file
830-831	Data communications trace file
832-833	Data communications configuration file
834-835	Data communications configuration database file
837	Sierra ZSYSCONF Config file
838	COUP IOP Configuration Database File
839	EMS formatter template (NLS capable)
840	SNAX utility output file

Table 2-2. System File Code Definitions (page 8 of 10)

File Code	Definition
841	COUP database file
842	COUP process image file
843	EMS logger file
844	EMS formatter template
845	EMS compiled filter
846	Cover file
847-848	NetBatch file
849	DNS configuration file
850	DNS database file
851	SNAX5 configuration file
852	NonStop CLX shutdown file
853-854	Optical disk file
855	FUP restart file
858	ORSERV status file
859	ODBC Catalog table
860	NSR millicode file
861	T16 only runnable object
862	Liberty only runnable object
863	SysHealth event alternate key file
870	Himalaya millicode file
880	RISC millicode file
881	NSK Disk and Tape Boots
882	SLSA Downloadable library file
888	Enform compiled query file
889	MMS Journal
890	ISDN Configuration File.
891	ISO FTAM
892	ISO FTAM
893-898	NETMASTER file
899	SNAX/XF ConfDef file
900	DSM/TC tape catalog file
901	SQL table filecode for DELPHI
904	Exchange trace file
941	SQL Collation object

Table 2-2. System File Code Definitions (page 9 of 10)

File Code	Definition
960	DSM/SCM (DELPHI) audit snapshot data
961	DSM/SCM (DELPHI) activation packages
962	DSM/SCM (DELPHI) dist file
963	DSM/SCM (DELPHI) SIT & INFO file
964	DSM/SCM (DELPHI) Audit information file
965	DSM/SCM (DELPHI) Software Archive file
966	DSM/SCM (DELPHI) "Binarized" data file
988	Trace file or an SCOMPRES archive file
990	Chameleon II Dump File
991	Not yet determined
992	NETMASTER help file
1000	SAFETNET key control file
1001	PRS TPR file
1002	Security card initialisation file
1003	PRS TPR file
1004	PRS TPR file
1012	PRS TPR file
1053-1057	PNA file
1100	SAFETNET key control file
1666	Transfer UOWTEST data file
1729	[TNSC] PAK/UNPAK file
2000	SAFETNET key control file or a DOCUSYS saved searchfile
2001	NSS database file
5000	TTSI TSMS file
5050	TTSI TSMS file
5101	TTSI TSMS file
5121	TTSI TSCP file
5140	QIO Control file
5201-5208	TTSI TSCP ENFORCE database file
5303	TTSI ERAD EMS crossref file
5304	TTSI ERAD definition file
5305	TTSI ERAD detail text file
5306	TTSI ERAD log file
5307	TTSI ERAD repair procedure file

Table 2-2. System File Code Definitions (page 10 of 10)

File Code	Definition
7878	TTSI Help file
9613	CRUNCH sysgen save file
9614	CPU dump file
11111	SEEVIEW trace file
19789	NSS Log file
22222	TTSI TSMS file

INFO Listing Format Example

This example shows the standard FUP INFO listing for four files on the current default subvolume and volume ($VOL1.SVOL).

The listing indicates that PARTFILE is a partitioned, key-sequenced file with an alternate key and that it has a record size of 80 bytes, block size of 1024 bytes, and 3072 bytes of data.

PARTFILE has file code 0 (represented by a blank field), was last modified on August 14, is owned by user 8,1, and has security “AO--”.

Example 2-3. INFO Listing Format

```
-INFO *FILE

            CODE  EOF  LAST MODIF     OWNER RWEP TYPE REC BL
$VOL1.SVOL
PARTFILE          3072 14AUG2000 14:55 8,1   AO--  PKA  80  1
MYFILE            5120 22JAN2001 11:44 8,44  AOAO  RA   10  1
ALTFILE           2048 22JAN2001 11:46 8,44  AOAO  K    11  1
NEWFILE    101    3206 22APR2001  9:01 8,4   ****
```

The four asterisks in the RWEP field indicate that the file NEWFILE is Safeguard protected.

Example 2-4. Short INFO for SQL/MX Table Using ANSI Names

```
FUP INFO 'TABLE CAT_ANSINAME01.SCH_ANSINAME01.TAB1 PARTITION
(PART1,PART2)'
            CODE  EOF  LAST MODIF     OWNER  RWEP TYPE REC  BL
$DATA05.ZSDHKPKT
CGJC4500  550A+ 12288 27Oct2005 23:34 -1  *SQL   PK   Ta  12  4
$DATA1.ZSDHKPKT
J1C73500  550A+ 12288 17Oct2005  7:32 -1  *SQL   XPK  Ta  12  4
```

Example 2-5. Short INFO for OSS Files With POSIX ACL

```
$SYSTEM SYSTEM 1> fup info
\OCTOPUS.$OSS.ZYQ00001.Z0000010
CODE EOF LAST MODIF OWNER RWEP TYPE REC BL
$OSS.ZYQ00001
Z0000010 OSS 0 13:48 -1 -rw-rw-rw-+
```

INFO DETAIL Listing Format

A DETAIL listing has two formats:

- One for SQL tables and indexes and for Enscribe and OSS files (For more information, see Example 2-6 on page 2-98.)
- One for SQL views

The information that appears depends on whether you are inquiring about a table, index, or file, and whether or not the organization is key-sequenced. Information is not shown if it is irrelevant or is not set for the specific file.

Example 2-6. DETAIL Format for SQL Tables and Indexes and for Enscribe and OSS Files

```
filename                                              date-and-time
1-  object-type
2-  CATALOG catalog-name
    VERSION number
    BASE TABLE base-table-name
3-  PHYSICAL FILENAME
    VIRTUAL FILENAME
    ANSI NAME ansi-name
    RESOURCE FORK resource-fork-location
    SYSTEM METADATA system-metadata-location
4-  TYPE file-type
5-  FORMAT format-code
6-  CODE file-code
7-  EXT ( pri-num PAGES, sec-num PAGES,
        MAXEXTENTS max-extents)
8-  REC record-length
    PACKED REC packed-record-length
    RECLENGTH max-record-length
    BLOCK block-length
9-  IBLOCK block-length
    KEY ( key-descriptor )
    SYSKEY
    LOCKLENGTH lock-length
    DCOMPRESS, ICOMPRESS
10- {INDEX } (key-spec, FILE alt-fnum, file-name,
    {ALTKEY} key-descriptor
               { UNIQUE | NO UNIQUE }
               , {UPDATE | NO UPDATE}, NULL null-value)        .
11- PART ( part-num , $volume , pri-ext PAGES,
         sec-ext PAGES, MAXEXTENTS max-ext,)
        firstkey-value        .
12- ODDUNSTR
    REFRESH
    AUDIT
    BUFFERSIZE
    BUFFERED
    AUDITCOMPRESS
    VERIFIEDWRITES
    SERIALWRITES
13- OWNER group-id,owner-id
    SECURITY (RWEP) : rwep, PROGID, CLEARONPURGE, LICENSE,
    TRUSTtrust-flag,
    (SUPPRESSED: rwep)
    NOPURGEUNTIL: expire-time
14- SECONDARY PARTITION
15- DATA MODIF: modif, open-state
    CREATION DATE: create-time
    REDEFINITION DATE: redefinition-time
    LAST OPEN: last-open-time
16- EOF eof(percent-used % USED)
17- FILE LABEL: num-bytes(percent-used % USED)
```

```
18- EXTENTS ALLOCATED: num-ext

19- INDEX LEVELS: num-index-levels
20- PARTITION ARRAY {EXTENDED | STANDARD | FORMAT2ENABLED}
```

Note. If you do not have access privileges to a file and you issue the INFO DETAIL command, UNAVAILABLE is displayed as the pathname.

In Example 2-6, DETAIL Format for SQL Tables and Indexes and for Enscribe and OSS Files, on page 2-98, the headers and variables are:

1. *object-type* indicates whether the file is an SQL/MP or SQL/MX base table, catalog table, index, or catalog index; Enscribe file; or Enscribe file containing an SQL/MP or SQL/MX object program.
   - INVALID indicates that an SQL/MP or SQL/MX object program is not valid and might need to be SQL/MP or SQL/MX compiled.
   - SHADOW LABEL indicates that the file is a shadow label. This file label exists temporarily after an SQL/MP or SQL/MX object is dropped and until the transaction is committed.
2. CATALOG identifies the catalog in which the object is defined. VERSION is the SQL/MX software version. (For more information about SQL/MX versioning, see the *SQL/MX Database and Application Migration Guide*.) If the file is an index, BASE TABLE is the underlying table.
3. PHYSICAL FILENAME indicates a logical file by its logical name. VIRTUAL FILENAME indicates a logical file by its physical name.
4. TYPE indicates the file organization:

   K Key sequenced

   E Entry sequenced

   R Relative

   U Unstructured
5. FORMAT is the new file's format designator, which can have these values:

Format Designator	Indicates the File Should Be a...
1	Format 1 file as described in Handling File Formats on page 1-22
2	Format 2 file as described in Handling File Formats on page 1-22

   If you omit the FORMAT option, the system decides the file format based on other file attributes.
6. CODE is the file code. File codes are displayed for SQL tables and indexes and for Enscribe files. The default file code of 0 is not displayed.

File codes in the range 100 through 999 refer to specific types of files and are reserved by HP. For a description of these codes, see Table 2-2, System File Code Definitions, on page 2-87.

Letters that follow the file code have specific meanings:

A TMF audits the file.

L The file is licensed by the super ID (255,255).

P The PROGID security attribute of the file is on.

7. Lists the sizes of the primary (*pri-num*) and secondary (*sec-num*) extents, and the maximum number of extents that can be allocated.
8. The items in this section do not appear for unstructured files:
   - REC indicates the maximum exploded record length for objects.
   - PACKED REC indicates the maximum packed record length for objects.
   - RECLENGTH indicates the maximum record length for relative tables.
   - BLOCK indicates the length of a block.
9. Describes the primary key of a key-sequenced file or other structured file type:
   - IBLOCK is the length of an index block of an Enscribe file.
   - KEY *key-descriptor* is one or more sets of these items (the number of sets is determined by the number of columns in the key):

     ```
     COLUMN col-num, OFFSET key-offset, TYPE col-type
        LENGTH key-length, {ASC }
                           {DESC}
     ```

     - COLUMN number indicates the position of the key column in the row. If the row contains a system-defined primary key, the primary key is column 0. Otherwise, the first column defined for the table is column 0.
     - OFFSET indicates the zero-relative byte address of the key column in the exploded record.
     - TYPE indicates the data type of the column. The data types supported by SQL/MX are a superset of the data types supported by SQL/MP.
     - LENGTH indicates the length of the key column in bytes.

     Note. The TYPE and LENGTH fields are not displayed for SQL/MX objects if SQL/MX returns default values (zeroes) for these fields.

     - ASC is ascending order, and DESC is descending order.
   - SYSKEY indicates a system-defined primary key.
   - LOCKLENGTH is the number of bytes of the primary key used for locking.
   - DCOMPRESS indicates keys in data blocks of the file are compressed.

- ICOMPRESS indicates keys in index blocks are compressed.

10. Describes indexes of an SQL table or alternate-key files of an Enscribe file:
    - The *key-spec* parameter is the key specifier stored in every index row.
    - FILE indicates by number an Enscribe alternate-key file.
    - The *key-descriptor* of the SQL index or Enscribe alternate-key file appears in the format for the KEY item (described in item 7).
    - UNIQUE (or NO UNIQUE) indicates whether a key is unique.
    - UPDATE (or NO UPDATE) indicates whether key specifiers of Enscribe files are automatically updated.
    - NULL indicates a null value set for an Enscribe file key.
11. Describes partitions if the object or file is partitioned. The partition name and volume name of each partition are followed by the number of primary and secondary extents and the maximum extent size allowed. The FIRST KEY value is given for a key-sequenced file. The partition information for SQL/MX objects is displayed without the partition key. The partition key information for SQL/MX objects is available from MXCI SHOWDDL.
12. Describes file attributes:

ODDUNSTR	An Enscribe odd unstructured file
REFRESH	The file label is updated when the file control block changes
AUDIT	An audited file
BUFFERSIZE	An unstructured DP2 file uses default internal transfer size
BUFFERED	Writes to file are buffered
AUDITCOMPRESS	Compressed audit-checkpoint messages are generated for DP2 files
VERIFYWRITES	Writes to file are verified
SERIALWRITES	Serial mirror writes are performed

13. OWNER is the user ID of the file's owner. This section also displays the security string of the file, which indicates whether the PROGID, TRUST, and CLEARONPURGE attributes are set, whether the LICENSE attribute is set, and when you can purge the file.

    TRUST *trust-flag* controls whether direct I/O access to user buffers is permitted when this process is running.

    NOPURGEUNTIL: *timestamp* (if included) indicates the expiration date set for a file. This is the date after which you can successfully purge the file.

    (SUPPRESSED: *rwep*) indicates the underlying security of a file protected at the file level by Safeguard. This indicates the security the file would have if Safeguard

security were removed. For more information, see the *Safeguard Reference Manual*.

14. SECONDARY PARTITION indicates the file is a secondary partition of an Enscribe file.

15. Lists dates and times of file activity. DATA MODIF indicates when the data in the file was last modified and one of the open states (if applicable). CREATION DATE indicates when the file was created. REDEFINITION TIME indicates when a change to the SQL table or index caused an SQL object program to be recompiled. LAST OPEN indicates when the file was last open.

    The modification date can be older than the file-creation date if the file was created by duplicating it with the FUP DUP command (with the SAVEALL or SOURCEDATE option).

    The open states are:

BLANK	The file is not open, failed, or broken.
BROKEN	The file is open but received an I/O or consistency check failure and needs media recovery.
CORRUPT	The file is corrupt. (The contents of the file are in question.) DUP and LOAD mark the destination files as corrupt while these operations are being performed. If the operation does not complete normally, the file is marked corrupt and should be purged.
DEFINITION INVALID	The data or definition of the object is invalid.
LABEL QUESTIONABLE	The file is in crash-label state. This state applies only to SQL views. A file is in the crash-label state if a file label operation was taking place at the time of a total system failure or if the disk on which it is located becomes unavailable.
OPEN	The file is open, or a TMF transaction is active on the file.
QUESTIONABLE	The file is in crash-open state. Either the file was open when a total system failure occurred, or the volume where the file resides became unavailable while the file was open.
REDO NEEDED	The file cannot be opened, and media recovery (redo) is needed.
UNDO NEEDED	The file cannot be opened, and media recovery (undo) is needed.

Note. For more information about media recovery, see the *TMF Operations and Recovery Guide*.

16. For unstructured files, EOF is the end-of-file pointer containing the relative byte address of the byte—following the last significant data byte.

    For structured files, EOF is the relative byte address of the first byte of the next available block.

    If all extents were allocated, the *percent-used* parameter is the amount of available file space currently used based on available space.

17. FILE LABEL is the number of bytes currently used for the file label and the percentage of the maximum file label it uses. If this is close to 100 percent full, the file cannot add any new extents.
18. EXTENTS ALLOCATED is the number of extents currently allocated for the file.
19. Indicates the number of index levels used for index blocks (for key-sequenced files).

Example 2-7 shows the format used by the FUP INFO command (with the DETAIL option) for SQL views:

Example 2-7. DETAIL Format for SQL/MP View

```
filename                        date-and-time
    object-type
    CATALOG catalog-name
    BASE TABLE base-table-name
    PART ( [ \node.] $volume )
         .
         .
    OWNER group-id,user-id
    SECURITY (RWEP): rwep
    LABEL QUESTIONABLE
    DEFINITION INVALID
    CREATION DATE: creation-date
    NOPURGEUNTIL: expire-time
    REDEFINITION DATE: redefinition-date
```

The DETAIL listing format for SQL/MP views contains the same type of information as the other DETAIL listing format. The BASE TABLE field indicates the name of the underlying table (for protection views), and the PART and REDEFINITION DATE fields also appear for protection views only. If the partition is on the current node, the node name does not appear. The LABEL QUESTIONABLE and DEFINITION INVALID fields are open states.

The DETAIL listing format for SQL/MX objects contains the same type of information as for SQL/MP objects except:

- *object-type* differentiates between different SQL/MX objects including ANSI tables, ANSI indexes, and metadata tables.
- *new-option* displays the location of the resource fork for an ANSI table, index, or metadata table.
- The file format is always 2.
- The security vector is ‘*SQL.’

The DETAIL listing format for SQL/MX objects also contains:

- The ANSI name associated with the object.

- The location of system metadata tables associated with the object.
- Clustering key information for the objects.
- Partitioning information for objects that can be partitioned.
- Indexes associated with tables.

INFO DETAIL Listing Format Examples

The listing examples in this section are for Enscribe files.

Note. For examples of listings for SQL files, see the *SQL/MP Reference Manual*.

- To show the FUP INFO, DETAIL listing for TEMPE (a DP2 entry-sequenced file):

```
-INFO TEMPE, DETAIL

$VOL1.SVOL.TEMPE            15 Apr 2001, 21:03
 ENSCRIBE
 TYPE E
 FORMAT 1
 EXT (1 PAGES, 1 PAGES)
 REC 80
 BLOCK  4096
 MAXEXTENTS 16
 OWNER 1,40
 SECURITY (RWEP): CUCU
 DATA MODIF:  14 Apr 2001, 16:59
 CREATION DATE:  10 Apr 1997, 16:00
 LAST OPEN:  14 Apr 2001, 18:00
 EOF 0 (0.0% USED)
 FILE LABEL: 214  (5.2 % USED)
 EXTENTS ALLOCATED: 0
```

- To show the FUP INFO, DETAIL listing for PARTFILE (a key-sequenced, partitioned file with alternate keys):

```
-INFO PARTFILE, DETAIL

$VOL1.SVOL.PARTFILE           11 Nov 2000, 14:16
 ENSCRIBE
 TYPE K
 FORMAT 1
 EXT ( 1 PAGES, 1 PAGES )
 REC 80
 BLOCK  4096
 IBLOCK 4096
 KEYLEN 10
 KEYOFF 0
 ALTKEY ( "ab", FILE 0, KEYOFF 10, KEYLEN 10 )
 ALTKEY ( "cd", FILE 1, KEYOFF 20, KEYLEN 10 )
 ALTFILE ( 0, $VOL1.SVOL.AK1 )
 ALTFILE ( 1, $VOL1.SVOL.AK2 )
 PART ( 1, $VOL2, 1, 1, "AA" )
 OWNER 8,1
```

```
 SECURITY (RWEP): AAAA
 DATA MODIF:  11 May 1999, 11:20
 CREATION DATE:  10 Apr 1997, 16:00
 LAST OPEN:  14 Apr 2000, 18:00
 EOF 3072  ( 9.4% USED)
 FILE LABEL: 298  (7.3 % USED)
 EXTENTS ALLOCATED: 2
 FREE BLOCKS 1
 INDEX LEVELS: 1
```

- To show the FUP INFO, DETAIL listing for a direct file (not a logical file):

```
-INFO $SYSTEM.SYS65.FUP, DETAIL

$SYSTEM.SYS65.FUP               3 Jul 2000, 16:50
 ENSCRIBE
 TYPE U
 CODE 100
 FORMAT 1
 EXT ( 228 PAGES, 64 PAGES )
 ODDUNSTR
 MAXEXTENTS 978
 BUFFERSIZE 4096
 OWNER -1
 SECURITY (RWEP): NONO, LICENSED
 DATA MODIF: 29 Apr 2000, 16:43, OPEN
 CREATION DATE: 2 Jul 1997, 14:10
 LAST OPEN: 3 Jul 2000, 16:48
 FILE LABEL: 342 (8.4% USED)
 EOF: 4909056 (3.8% USED)
 EXTENTS ALLOCATED: 35
```

- To show the FUP INFO, DETAIL listing for a logical file by its logical name:

```
-INFO $BALL.CAROLS.FILE, DETAIL

$BALL.CAROLS.FILE               3 Jul 2000, 16:49
 ENSCRIBE
 PHYSICAL FILENAME: $HEAT.ZYS00000.A00057I0
 TYPE U
 FORMAT 1
 EXT ( 2 PAGES, 2 PAGES )
 MAXEXTENTS 16
 BUFFERSIZE 4096
 OWNER -1
 SECURITY (RWEP): NUNU
 DATA MODIF: 3 Jul 2000, 16:48
 CREATION DATE: 3 Jul 2000, 14:48
 LAST OPEN: NEVER OPENED
 FILE LABEL: 314 (7.7% USED)
 EOF: 0 (0.0% USED)
 EXTENTS ALLOCATED: 0
```

- To show the FUP INFO, DETAIL listing for a logical file by its physical name:

```
-INFO $HEAT.ZYS00000.Z00057I0, DETAIL

$HEAT.ZYS00000.A00057I0                3 Jul 2000, 16:50
 ENSCRIBE
 VIRTUAL FILENAME: \SMSDEV.$BALL.CAROLS.FILE
 TYPE U
 FORMAT 1
 EXT ( 2 PAGES, 2 PAGES )
 MAXEXTENTS 16
 BUFFERSIZE 4096
 OWNER -1
 SECURITY (RWEP): NUNU
 DATA MODIF: 3 Jul 2000, 16:48
 CREATION DATE: 3 Jul 2000, 14:48
 LAST OPEN: NEVER OPENED
 FILE LABEL: 314 (7.7% USED)
 EOF: 0 (0.0% USED)
 EXTENTS ALLOCATED: 0
```

- File attributes are set using the SET command in this example. The SHOW command displays the results of the attributes assigned by the previous SET commands:

```
-SET TYPE K
-SET KEYLEN 2
-SET ALTKEY ("AA",FILE 0,KEYLEN 2,KEYOFF 0,INSERTIONORDER)
-SET ALTFILE (0, ALT0)
-SHOW TYPE K
 FORMAT 1
 EXT (1 PAGES, 1 PAGES)
 REC 80
 BLOCK 4096
 IBLOCK 4096
 KEYLEN 2
 KEYOFF 0
 ALTKEY ("AA", FILE 0, KEYOFF 0, KEYLEN 2, INSERTIONORDER)
 ALTFILE ( 0, $DATAA.DCDTEST.ALT0 )
 ALTCREATE
 MAXEXTENTS 16
 -
```

Next, issue a CREATE KEY command. (FUP responds by displaying information on the keys just created.) Then issue an INFO KEY, DETAIL command:

```
-CREATE KEY
CREATED - $DATAA.DCDTEST.KEY
CREATED - $DATAA.DCDTEST.ALT0

-INFO KEY, DETAIL

$DATAA.DCDTEST.KEY         17 Dec 2000, 16:45
 ENSCRIBE
 TYPE K
 FORMAT 1
 EXT (2 PAGES, 2 PAGES)
```

```
 REC 80
 BLOCK 4096
 IBLOCK 4096
 KEYLEN 2
 KEYOFF 0
 ALTKEY ("AA",FILE 0,KEYOFF 0,KEYLEN 2,INSERTIONORDER)
 ALTFILE ( 0, $DATAA.DCDTEST.ALT0 )
 MAXEXTENTS 16
 OWNER 1, 164
 SECURITY (RWEP): CUCU
 DATA MODIF:  17 Dec 2000, 16:45
 CREATION DATE:  17 Dec 1997, 16:45
 LAST OPEN: NEVER OPENED
 EOF 0 (0.0% USED)
 FILE LABEL: 248  (6.1 % USED)
 EXTENTS ALLOCATED: 0
 INDEX LEVELS: 0
```

- To request detailed information for a Format 2 partitioned SQL/MP table:

```
-INFO T2, DETAIL
 $DATA00.TEST.T2       19 May 2003, 17:08
 SQL BASE TABLE
 CATALOG $DATA00.TEST
 VERSION 350
 TYPE K
 FORMAT 1
 EXT ( 16 PAGES, 64 PAGES, MAXEXTENTS 160 )
 REC 46
 PACKED REC 46
 BLOCK 4096
 KEY ( COLUMN 0, OFFSET 0, LENGTH 4, ASC )
 PART ( 0, $DATA00, 16 PAGES, 64 PAGES, MAXEXTENTS 160,
 FORMAT 1, -2147483648 )
 PART ( 1, $D72GB1, 16 PAGES, 64 PAGES, MAXEXTENTS 160,
 FORMAT  1, 40 )
 AUDIT
 BUFFERED
 AUDITCOMPRESS
 OWNER -1
 SECURITY (RWEP): NUNU
 DATA MODIF:  19 May 2003, 15:25
 CREATION DATE:  19 May 2003, 15:25
 REDEFINITION DATE:  19 May 2003, 15:24
 LAST OPEN: NEVER OPENED
 EOF: 0 (0.0% USED)
 EXTENTS ALLOCATED: 0
 INDEX LEVELS: 0
 PARTITION ARRAY FORMAT2ENABLED
```

- To request detailed information about ALT0:

```
-INFO ALT0, DETAIL

$DATAA.DCDTEST.ALT0         17 Dec 2000, 16:46
 ENSCRIBE
```

```
 TYPE K
 FORMAT 1
 EXT (2 PAGES, 2 PAGES )
 REC 14
 BLOCK 4096
 IBLOCK 4096
 KEYLEN 14
 KEYOFF 0
 MAXEXTENTS 16
 OWNER 1,164
 SECURITY (RWEP): CUCU
 DATA MODIF:  17 Dec 2000, 16:45
 CREATION DATE:  17 Dec 1997, 16:45
 LAST OPEN:  NEVER OPENED
 EOF 0 (0.0% USED)
 FILE LABEL: 214  (5.2 % USED)
 EXTENTS ALLOCATED: 0
 INDEX LEVELS: 0
```

- To request detailed information about all subvolumes that begin with ER:

```
- INFO ER*.*, DETAIL

$GUEST.ERIC.TACLCSTM                 13 Aug 2000, 14:04
 ENSCRIBE
 TYPE U
 CODE 101
 FORMAT 1
 EXT ( 4 PAGES, 16 PAGES )
 MAXEXTENTS 16
 BUFFERSIZE 4096
 OWNER 96,27
 SECURITY (RWEP): NUNU
 DATA MODIF:  22 Jun 1999, 10:25
 CREATION DATE:  22 Jun 1997, 10:25
 LAST OPEN:   7 May 2000, 15:32
 EOF 80 (0.0% USED)
 FILE LABEL: 214  (5.2 % USED)
 EXTENTS ALLOCATED:  1

$GUEST.ERD.A                         13 Aug 2000, 14:04
 ENSCRIBE
 TYPE U
 CODE 101
 FORMAT 1
 EXT ( 2 PAGES, 2 PAGES )
 MAXEXTENTS 16
 BUFFERSIZE 4096
 OWNER 64,5
 SECURITY (RWEP): AAAA
 DATA MODIF:  27 Jun 2000, 11:04
 CREATION DATE:  27 Jun 1997, 11:03
 LAST OPEN:  25 Jul 2000, 18:37
 EOF 40 (0.1% USED)
```

```
 FILE LABEL: 214  (5.2 % USED)
 EXTENTS ALLOCATED:  1

$GUEST.ERNIE.ABEND                    13 Aug 2000, 14:04
 ENSCRIBE
 TYPE U
 CODE 100
 FORMAT 1
 EXT ( 4 PAGES, 2 PAGES )
 ODDUNSTR
 MAXEXTENTS 16
 BUFFERSIZE 4096
 BUFFERED
 OWNER 64,5
 SECURITY (RWEP): UUUU
 DATA MODIF:  28 Jun 2000, 15:34
 CREATION DATE:  28 Jun 2000, 15:34
 LAST OPEN:  28 Jun 2000, 16:00
 EOF 7116 (10.2% USED)
 FILE LABEL: 214  (5.2 % USED)
 EXTENTS ALLOCATED:  1
```

- To request detailed information for a partitioned SQL/MX table:

```
-INFO IIP3BN00, DETAIL

$DATA04.ZSD4897J.IIP3BN00             2 Sep 2003,  4:36
 SQL ANSI TABLE
 ANSI NAME CAT.SCH.SACHIN_JOHN
 RESOURCE FORK \APACHE.$DATA04.ZSD4897J.IIP3BN01
 SYSTEM METADATA \APACHE.$DATA04.ZSD0JUL
 VERSION 1200
 TYPE K
 FORMAT 2
 CODE 550
 EXT ( 16 PAGES, 64 PAGES, MAXEXTENTS 160 )
 PACKED REC 8
 BLOCK 4096
 KEY ( COLUMN 0, TYPE 0, LENGTH 0, ASC  )
 PART ( 0, \APACHE.$DATA04.ZSD4897J.IIP3BN00 )
 PART ( 1, \APACHE.$DATA04.ZSD12345.PARAS100 )
 AUDIT
 BUFFERED
 AUDITCOMPRESS
 OWNER -1
 SECURITY (RWEP): *SQL
 DATA MODIF:  26 Aug 2003, 22:36
 CREATION DATE:  26 Aug 2003, 22:36
 REDEFINITION DATE:  26 Aug 2003, 22:36
 LAST OPEN:   2 Sep 2003,  4:24
 EOF: 0 (0.0% USED)
 EXTENTS ALLOCATED: 0
 INDEX LEVELS: 0
 PARTITION ARRAY FORMAT2ENABLED
```

- To request detailed information for a partitioned SQL/MX table using ANSI names:

```
FUP INFO 'TABLE CAT_ANSINAME01.SCH_ANSINAME01.TAB1',DETAIL

$DATA05.ZSDHKPKT.CGJC4500            16 Nov 2005,  0:38

    SQL ANSI TABLE
    ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
    RESOURCE FORK \DRP42.$DATA05.ZSDHKPKT.CGJC4501
    SYSTEM METADATA \DRP42.$DATA05.ZSD0
    VERSION 1200
    TYPE K
    FORMAT 2
    CODE 550
    EXT ( 16 PAGES, 64 PAGES, MAXEXTENTS 160 )
    PACKED REC 12
    BLOCK 4096
    KEY ( COLUMN 0, ASC  )
    INDEX ( 0, \DRP42.$DATA05.ZSDHKPKT.DK8CK600 )
    PART ( 0, \DRP42.$DATA05.ZSDHKPKT.CGJC4500 )
    PART ( 1, \DRP42.$DATA1.ZSDHKPKT.J1C73500 )
    PART ( 2, \DRP42.$DATA2.ZSDHKPKT.KD973500 )
    PART ( 3, \DRP42.$DATA3.ZSDHKPKT.L9G93500 )
    PART ( 4, \DRP42.$DATA4.ZSDHKPKT.NQW83500 )
    PART ( 5, \DRP42.$DATA05.ZSDHKPKT.P1G73500 )
    PART ( 6, \DRP42.$DATA1.ZSDHKPKT.PJ183500 )
    PART ( 7, \DRP42.$DATA2.ZSDHKPKT.Q6KB4500 )
    PART ( 8, \DRP42.$DATA3.ZSDHKPKT.SVS83500 )
    PART ( 9, \DRP42.$DATA4.ZSDHKPKT.WTK93500 )
    AUDIT
    BUFFERED
    AUDITCOMPRESS
    OWNER -1
    SECURITY (RWEP): *SQL
    DATA MODIF:  27 Oct 2005, 23:34
    CREATION DATE:  16 Oct 2005, 20:31
    REDEFINITION DATE:  16 Oct 2005, 20:31
    LAST OPEN:  16 Nov 2005,  0:33
    EOF: 12288 (0.1% USED)
    EXTENTS ALLOCATED: 1
    INDEX LEVELS: 1
    PARTITION ARRAY FORMAT2ENABLED
```

- To request detailed information for an OSS File with POSIX ACL:

```
$SYSTEM SYSTEM 2> fup info \OCTOPUS.$OSS.ZYQ00001.Z0000010,
detail
$SYSTEM SYSTEM 2..
$OSS.ZYQ00001.Z0000010 11 May 2006, 13:59 OSS
PATH: /aclutils/file1
OWNER -1
SECURITY: -rw-rw-rw-+
CREATION DATE: 11 May 2006, 13:48
ACCESS TIME: 11 May 2006, 13:48
EOF: 0
```

INFO STATISTICS Listing Format

Example 2-8 shows the format that the FUP INFO command (with the STATISTICS option) uses to display file information. It reads the specified file to gather the statistics and requires more time to complete than the other INFO commands.

You must have read-access to the file. STATISTICS information is not listed for unstructured or entry-sequenced files.

Example 2-8. INFO STATISTICS Listing Format

```
            TOTAL     TOTAL    AVG #     AVG       AVG %
 LEVEL     BLOCKS     RECS     RECS     SLACK      SLACK     [ PART ]
 level   t-blocks   t-recs   a-recs   a-slack   a-%-slack   [ name ]
   .         .         .        .         .         .           .
   .         .         .        .         .         .           .
[ FREE   t-blocks          ]
[ FREE               t-recs ]
[ BITMAP t-blocks ]
```

`level`

indicates the tree level of the entry. Values for *`level`* are:

DATA Indicates that the entry is for the data level. Only this level is shown for relative and entry-sequenced files.

A One or greater indicates an index level, and one (1) is the lowest. Index levels are shown only for key-sequenced files.

`t-blocks`

is the total number of blocks in use at the indicated level.

`t-recs`

is the total number of records at the indicated level. At the DATA level, *`t-recs`* is the total number of data records in the file.

`a-recs`

is the average number of records for each block at the indicated level.

`a-slack`

is the average number of unused bytes for each block at the indicated level.

`a-%-slack`

is the average percentage of unused bytes for each block at the indicated level.

name

is shown only if the file has extra partitions. It is the volume name of the partition associated with the entry.

FREE *t-blocks*

is the total number of unused blocks in the file between the beginning of the file and the current EOF (for key-sequenced files).

FREE *t-recs*

is the total number of empty records in the file between the beginning of the file and the current EOF location (for relative files).

BITMAP *t-blocks*

is the number of bitmap blocks (for DP2 relative and key-sequenced files only).

INFO STATISTICS Listing Format Examples

- To display the FUP INFO, STATISTICS listing for a key-sequenced, partitioned file:

```
-INFO PARTFILE,STATISTICS
(DETAIL option listing displays first, followed by this)
       TOTAL     TOTAL  AVG #   AVG   AVG %
LEVEL  BLOCKS     RECS   RECS  SLACK  SLACK  PART
  1        1        1    1.0   996     97    $VOL1
 DATA      1       12   12.0   338     33
 FREE      1
  1        1        7    7.0   938     92    $VOL2
 DATA      7      117   16.7    77      8
 FREE      1
```

- To display the FUP INFO, STATISTICS listing for a key-sequenced, partitioned file with 48989 records and an EOF of 6983680:

```
-INFO SPECIALK,STAT
$GRAIN.CEREAL.SPECIALK        17 Feb 2001, 11:40
       TOTAL     TOTAL  AVG #   AVG   AVG %
LEVEL  BLOCKS     RECS   RECS  SLACK  SLACK
  2        1       14   14.0  3760     92
  1       14     1689  120.6  1113     27
 DATA   1689    48989   29.0  1414     35
 FREE      0
 BITMAP    1
```

- To display the FUP INFO, STATISTICS, PARTIAL listing for a key-sequenced, partitioned file with 48989 records and an EOF of 6983680:

```
-INFO SPECIALK,STAT,PARTIAL 10
$GRAIN.CEREAL.SPECIALK        17 Feb 2001, 11:44
      *****PARTIAL STATISTICS:  10% OF FILE*****
                          AVG #   AVG   AVG %
LEVEL  BLOCKS     RECS   RECS  SLACK  SLACK
```

```
  2        1      14   14.0  3760     92
  1        1      75   75.0  2054     50
 DATA    166    4840   29.2  1363     33
 FREE      0
 BITMAP    1
```

- To display the FUP INFO, STATISTICS, PARTIAL listing for a partitioned SQL/MX table:

```
 -INFO WWVL3T00, STAT, PARTIAL 30
(DETAIL option listing displays first, followed by this)
***** PARTIAL STATISTICS: 30% OF FILE *****
                          AVG #   AVG  AVG %
LEVEL   BLOCKS     RECS   RECS  SLACK SLACK  PART
  1         1         1    1.0   4036    99
$D1103.ZSDL2BDF.WWVL3T00
DATA        1        19   19.0   3170    77
FREE        0
BITMAP      1
  1         1         3    3.0   4012    98
$D1103.ZSDL2BDF.J4XL3T00
DATA        3       179   59.7   1299    32
FREE        0
BITMAP      1
```

- To display the FUP INFO, STATISTICS, PARTONLY listing for a partitioned SQL/MX table using ANSI name:

```
-INFO 'TABLE
CAT_ANSINAME01.SCH_ANSINAME01.TAB1',STAT,PARTONLY

$DATA05.ZSDHKPKT.CGJC4500            16 Nov 2005,  0:38
  SQL ANSI TABLE
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  RESOURCE FORK \DRP42.$DATA05.ZSDHKPKT.CGJC4501
  SYSTEM METADATA \DRP42.$DATA05.ZSD0
  VERSION 1200
  TYPE K
  FORMAT 2
  CODE 550
  EXT ( 16 PAGES, 64 PAGES, MAXEXTENTS 160 )
  PACKED REC 12
  BLOCK 4096
  KEY ( COLUMN 0, ASC  )
  INDEX ( 0, \DRP42.$DATA05.ZSDHKPKT.DK8CK600 )
  PART ( 0, \DRP42.$DATA05.ZSDHKPKT.CGJC4500 )
  PART ( 1, \DRP42.$DATA1.ZSDHKPKT.J1C73500 )
  PART ( 2, \DRP42.$DATA2.ZSDHKPKT.KD973500 )
  PART ( 3, \DRP42.$DATA3.ZSDHKPKT.L9G93500 )
  PART ( 4, \DRP42.$DATA4.ZSDHKPKT.NQW83500 )
  PART ( 5, \DRP42.$DATA05.ZSDHKPKT.P1G73500 )
  PART ( 6, \DRP42.$DATA1.ZSDHKPKT.PJ183500 )
  PART ( 7, \DRP42.$DATA2.ZSDHKPKT.Q6KB4500 )
  PART ( 8, \DRP42.$DATA3.ZSDHKPKT.SVS83500 )
  PART ( 9, \DRP42.$DATA4.ZSDHKPKT.WTK93500 )
  AUDIT
```

```
BUFFERED
AUDITCOMPRESS
OWNER -1
SECURITY (RWEP): *SQL
DATA MODIF:  27 Oct 2005, 23:34
CREATION DATE:  16 Oct 2005, 20:31
REDEFINITION DATE:  16 Oct 2005, 20:31
LAST OPEN:  16 Nov 2005,  0:38
EOF: 12288 (0.1% USED)
EXTENTS ALLOCATED: 1
PARTITION ARRAY FORMAT2ENABLED
TOTAL     TOTAL  AVG #   AVG   AVG %
LEVEL    BLOCKS     RECS   RECS  SLACK  SLACK
1           1         1    1.0   4036     99
DATA        1        29   29.0   3580     87
FREE        0
BITMAP      1
```

INFO EXTENTS Listing Format

Example 2-9 shows the format the FUP INFO command with the EXTENTS option uses to display file information.

Example 2-9. INFO EXTENTS Listing Format

```
filename                                date-and-time
    EXTENT     # OF PAGES   STARTING PAGE [  PART  ]
    extent-num num-pages    start-page    [  name  ]
```

filename	Is the name of the file being listed by FUP INFO.
date-and-time	Indicates the system date and time when the listing was produced.
extent-num	Is the ordinal extent number of the entry. The first extent in a file is designated extent 0. If no extents are allocated, the value is NONE.
num-pages	Is the number of disk pages (2048-byte units) that compose the indicated extent.
start-page	Is the absolute page address of the first page of the indicated extent.
name	For partitioned files, is the partition name associated with the entry.

INFO EXTENTS Listing Format Examples

- To show the FUP INFO, EXTENTS listing for PARTFILE (a key-sequenced, partitioned file):

```
-INFO PARTFILE,EXTENTS
$VOL1.SVOL.PARTFILE           11/03/99 14:45
      EXTENT  # OF PAGES  STARTING PAGE  PART
          0           1         12246    $VOL1
          1           1         12247
```

```
0           1           239    $VOL2
1           1           293
2           1           294
3           1           297
4           1           307
```

- To show the FUP INFO, EXTENTS listing for a partitioned SQL/MX table:

```
-INFO J4XL3T00, EXTENTS
$D1103.ZSDL2BDF.J4XL3T00              27 Jan 2004, 13:39
  EXTENT  # OF PAGES  STARTING PAGE  PART

     0         16       176539     \SURYA.$D1103.ZSDL2BDF.WWVL3T00
     0         16       167595     \SURYA.$D1103.ZSDL2BDF.J4XL3T00
```

- To show FUP INFO, EXTENTS listing for a partitioned SQL/MX table using ANSI names:

```
-INFO 'TABLE CAT_ANSINAME01.SCH_ANSINAME01.TAB1',EXTENTS

$DATA05.ZSDHKPKT.CGJC4500             16 Nov 2005,  0:38
EXTENT # OF PAGES STARTING PAGE  PART

0      16    2020385     \DRP42.$DATA05.ZSDHKPKT.CGJC4500
0      16    5728688     \DRP42.$DATA1.ZSDHKPKT.J1C73500
0      16    336363      \DRP42.$DATA2.ZSDHKPKT.KD973500
0      16    15263976    \DRP42.$DATA3.ZSDHKPKT.L9G93500
0      16    2356668     \DRP42.$DATA4.ZSDHKPKT.NQW83500
0      16    2020401     \DRP42.$DATA05.ZSDHKPKT.P1G73500
0      16    5728736     \DRP42.$DATA1.ZSDHKPKT.PJ183500
0      16    364775      \DRP42.$DATA2.ZSDHKPKT.Q6KB4500
0      16    15263928    \DRP42.$DATA3.ZSDHKPKT.SVS83500
0      16    2356636     \DRP42.$DATA4.ZSDHKPKT.WTK93500
```

Commands Related to INFO

COMMAND	Function	Page
FILES	Displays the names of all files in a subvolume	2-74
FILENAMES	Displays the names of files	2-72
CONFIG[URE]	Sets default option for STAT and STATONLY	2-26

LICENSE (Super ID)

Lets nonprivileged users execute TAL programs that contain privileged attributes (CALLABLE or PRIV attributes). Only a super ID (255,255) user can use this command. This command applies only to Enscribe files.

Note. For more information about licensing programs, see the *Guardian User's Guide .*

Only the super ID can run a privileged program that is not licensed, run license privileged programs, or use the REVOKE command to revoke the license of a

privileged program. To license files protected by the Safeguard product, use the Safeguard command interpreter (SAFECOM).

```
LICENSE fileset-list
```

fileset-list

is a list of files to be licensed for use by nonprivileged users. You can use wild-card characters and specify *qualified-fileset* for *fileset-list*.

LICENSE (Super ID) Guidelines

- Files must be code 100, 700, or 800. If a file has another file code, error 2 occurs.
- If a licensed, privileged program is opened with write access, the file becomes unlicensed.
- LICENSE works only with SQL files that are not SQL object files.
- If a user who is not the super ID (255,255) attempts to license a file, file-system error 48 (security violation) occurs.

LICENSE (Super ID) Examples

To let nonprivileged users run the privileged program stored in the disk file MYPROG (if the SUPER ID enters the command:

```
-LICENSE MYPROG
```

To let nonprivileged users run any of the privileged programs stored in the current subvolume with a file code of 100 (if the super ID enters the command):

```
-LICENSE * WHERE FILECODE=100
```

Commands Related to LICENSE (Super ID)

COMMAND	Function	Page
REVOKE	Resets file security and other attributes of a file	2-159

LISTLOCKS

Displays information on all locks (granted or waiting) for specified Guardian file sets or SQL/MX ANSI names.

FUP converts each ANSI name to the corresponding list of Guardian file names, then performs LISTLOCKS on each of these files, and displays information of the locked files only.

Note. FUP support for fully qualified ANSI names for the LISTLOCKS command is applicable on H06.04 and subsequent RVUs.

By default, locks on all partitions of a partitioned Enscribe file are displayed. Use LISTLOCKS to clarify lock situations. This command does not provide instantaneous views of the locks.

```
LISTLOCKS  [ / OUT listfile / ] fileset-list / ansiname-list
           [ , GRANTED ] [ , DETAIL ] [ , PARTONLY ]
```

OUT *listfile*

names a file or device to receive the listing output of the LISTLOCKS command. You can use either a standard file name or a spool DEFINE name as the OUT *listfile*. If *listfile* is an existing file, FUP appends output to it.

Note. For more information, see Specifying Files on page 1-8.

fileset-list

is a set of files for which to list existing locks (including SQL files and SQL object files). You can use wild-card characters and can specify *qualified-fileset*.

ansiname-list

ansiname-list = 'ansiname' [, 'ansiname']...

identifies SQL/MX ANSI Name catalogs, schemas, tables, indexes, partitions of tables and indexes, and any combination of these objects. A single quote (') is required to precede and delimit each ansiname. The ANSI names syntax is in accordance with Unified Syntax Proposal. The syntax is:

```
ansiname :: = CATALOG  [SYSTEM  |  USER ] {* | SQL-names}|
              SCHEMA  [SYSTEM ]  SQL-names |
               {TABLE  |  INDEX}  base-mx-object-names

SQL-names  :: =  SQL-name |( SQL-name  [, SQL-name … ]... )

base-mx-object-names  :: =  base-mx-object-name            |
              ( base-mx-object-name [, base-mx-object-name …] )

base-mx-object-name ::= SQL-name [ partitions ]
partitions  :: = PARTITION (SQL-identifier  [, SQL-identifier
…]...)
```

SQL-name

is used to name SQL base objects (such as tables or indexes) in addition to their SQL containers: catalogs and schemas. The names (called 3-part names) for SQL base objects such as tables, indexes, or modules are composed of three SQL identifiers separated by two dot characters (for example, CAT.SCH.T).

SQL-identifier

is a name used by SQL/MX to identify tables, views, columns, and other SQL entities. SQL identifiers can be either regular or delimited and can contain up to 258 characters in external form, or equivalently up to 128 characters in internal format. Regular identifiers begin with a letter (A through Z or a through z), but can also contain digits (0 through 9), or underscore characters (_).

Regular identifiers used to name a SQL/MX module (the basic object part) can start with the ^ character or contain the ^ character.

Note. The information regarding SQL/MX module provided above is for reference purpose only. FUP commands do not support the MODULE keyword.

A delimited identifier is enclosed in double quotes ("). Delimited identifiers are character strings that appear within double quote characters (") and consist of alphanumeric characters and other characters, except for character @, /,\, and ^. To include a double quote character in a delimited identifier, use two consecutive double quotes. A delimited module name in SQL/MX can contain the circumflex character (^).

Note. The keywords, SYSTEM and USER, help distinguish user data from metadata. The SYSTEM keyword can be used together only with the keyword CATALOG or SCHEMA, to indicate the system metadata contained inside a catalog or schema. The USER keyword can only be used with the CATALOG keyword to indicate the user metadata contained inside a catalog. The set of tables defined by USER and those defined by SYSTEM are mutually exclusive. The CATALOG or SCHEMA keyword without the SYSTEM keyword, or the CATALOG keyword without the USER keyword, indicates both the user data and the metadata.

The SCHEMA USER keyword is not supported and FUP returns an error if the parsed ANSI name is of this type.

`GRANTED`

specifies to list only currently granted locks. Locks in a waiting state are not shown.

`DETAIL`

specifies that the internal LOCK STATE is displayed.

`PARTONLY`

specifies to list only the locks against the specified partition.

LISTLOCKS Listing Format

Example 2-10 shows the format the FUP LISTLOCKS command uses to display file information. The first line lists the file name specified in the LISTLOCKS command.

Example 2-10. FUP LISTLOCKS DETAIL Listing Format

```
\node.$volume.subvolume.file-id

  LOCK          REQUESTER                    KEY
  TYPE  STATE   ID                           LEN   KEY/RECORD ADDRESS

  F     G       \FOXII.$:3:52:17484380

  LOCK STATE = LK^X
```

LOCK TYPE

is the type of lock:

F	File
R	Record
RG	Generic record

STATE

is the state of the lock:

G	Granted
I	Internally generated intent lock
W	Waiting

REQUESTER ID

is either a named or unnamed process (or a transaction ID).

KEY LEN

is the key length.

KEY/RECORD ADDRESS

is a key value for each locked record according to the type of file:

- For key-sequenced files, the key value is displayed. This value can wrap around to the next line.
- For unstructured files, the relative byte address is displayed.
- For entry-sequenced files, the record address is displayed.
- For relative record files, the record number is displayed. The key field cannot be longer than the generic lock length for generic key locks.

LOCK STATE

specifies the size and range of the lock.

LK^IS	Intent shared. This lock is acquired for the table only. You can upgrade it to LK^S when escalation to a table lock is required.
LK^IX	Intent exclusive. This lock is acquired for the table only. You can upgrade it to LK^X when escalation to a table lock is required.
LK^R	Range check. This lock assures that the range is not protected by another lock before an insert. It is always released after it is granted.
LK^US	Unique shared. This lock is acquired to protect a single row (no range protection).
LK^S	Shared. This lock protects an entire table if granted for the table. Otherwise it protects the row that is locked and the range between the locked row and the row that precedes it.
LK^D	Delete. This lock is acquired for the row after a deleted row. Use it to prevent scans from skipping uncommitted deletes.
LKDUS	Delete. This lock is acquired for the row after a deleted row. Use it to prevent scans from skipping uncommitted deletes.
LKDS	Delete shared. This lock is a composite of LK^D and LK^S. It is a shared lock for the row and a delete of one or more of the rows that precede the locked row.
LKSIX	Shared, intent exclusive. This lock is acquired for a table when an LK^IX table lock exists and a user duration LK^S table lock is required for a scan.
LK^UX	Unique, exclusive. This lock is acquired to protect a single row (no range protection).
LKDUX	Delete, unique exclusive. This lock is a composite of LK^D and LK^UX. It describes a unique exclusive lock for the row and a delete of one or more rows before the locked row.
LK^X	Exclusive. This lock is acquired to provide exclusive protection for the row that is locked and for the range between the locked row and the row before it.
LK^DP2	DP2 key-link (either left or right key linklock). This is not a lock but is maintained in the same manner. A key-link is an entity that helps DP2 process efficiency and is in the output of most utilities that display lock information.

LISTLOCKS Guidelines

- Use the REPORTWIDTH option (from the CONFIGURE command) to set the maximum length (in columns) for a subsequent LISTLOCKS listing.
- The LISTLOCKS display for SQL files sometimes shows meaningless values in one row; for example, a row of the number 255 (or all zeros).

- An intent lock is internally generated by a disk process to control combinations of record and file locks on a file. When a disk process receives a requester for a record lock, it requests and receives an intent lock on the file. If the requested record is not already locked, the record lock is granted.

  An intent lock indicates that the holder has (or intends to have) one or more records locked in a file. If a file lock has been issued against a file, intent locks are not granted—but multiple intent locks can be granted against the same file.

- A single LISTLOCKS report can contain inconsistencies (including record locks that have no corresponding intent lock) because the status of locks can change while the LISTLOCKS output is being prepared.
- If a file has multiple partitions, the file (F) lock indicates a file lock against that partition, not all partitions.
- The LISTLOCKS command returns Error 48 when it is executed against a file on a remote node and an incompatibility exists between the product version of FUP and DP2 on the network. Execute the LISTLOCKS command from a copy of FUP running on the remote node.
- If you specify the primary partition of an Enscribe partitioned file, LISTLOCKS displays lock information (if any) for all the partitions of the file.
- The user running FUP must have remote access to any system, which is implicitly referenced by the ANSI name used in the FUP command. For example, if the ANSI name is 'CATALOG*', the user must have access to any node on which any visible catalogs reside, and so on. If the ANSI name is 'TABLE C.S.T', the user must have access to any node on which partitions of table C.S.T reside, and so on. Users who want to limit the scope of the command to all SQL/MX objects on the local machine can use a Guardian wildcard of the form: $*.ZSD*.*.
- The above explanation can be used to explain both an error 8551 from ANSI names or error 48 from the file system, depending on the command that was used.
- LISTLOCK supports SQL/MX objects, CATALOG, SCHEMA, TABLE, INDEX, and PARTITIONS.

LISTLOCKS Example

- To display all locks on files contained in subvolume $DATA.SUBVOL that end with FILE:

  ```
  -LISTLOCKS $DATA.SUBVOL.*FILE
  ```

  FUP displays the locks listed for two files (KSFILE and ENTFILE):

  ```
  \MYSYS.$DATA.SUBVOL.KSFILE
   LOCK          REQUESTER           KEY
   TYPE   STATE  ID                  LEN  KEY/RECORD ADDRESS
   R      G      \MYSYS.02,132       10  "AT1200-659"
   R      G      \MYSYS.$MYPROG       5  "Z0072"
   R      W      \MYSYS.01,044        5  "Z0072"
  ```

```
 R     G      \MYSYS(1):14:3342   12  "values" ?0 ?11 ?6


\MYSYS.$DATA.SUBVOL.ENTFILE
 LOCK          REQUESTER            KEY
 TYPE  STATE   ID                   LEN  KEY/RECORD ADDRESS
 R     G       \MYSYS.$MYPROG             477184
 R     G       \MYSYS(162):15:002314     2502656
 R     W       \MYSYS.01,044            11264000
 F     WI      \MYSYS.$MYPROG
```

The locks in this example are grouped by locked resource. A locked resource occurs when the locks on a file are grouped together, and the locks on a record in that file are also grouped together.

When the file-set list is a single full volume (for example, $*volume*.*.*), a volume request is made to the Guardian file system to retrieve the lock information. The lock information is retrieved quicker (but unsorted), and FUP displays it in unsorted order.

- To display locks on SQL/MX index using ANSI names:

```
FUP LISTLOCKS 'INDEX CAT_ANSINAME01.SCH_ANSINAME01.IND1
PARTITION (IPART1)',DETAIL

$DATA05.ZSDHKPKT.DK8CK600
ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.IND1
LOCK          REQUESTER               KEY

TYPE  STATE  ID                      LEN  KEY/RECORD ADDRESS
F      GI    \DRP42(2).2.7784507
LOCK STATE  = LK^IX
R     G      \DRP42(2).2.7784507  8 ?0 ?0 ?0 ?12 ?0 ?0 ?0 ?11
LOCK STATE = LK^UX
```

LISTOPENS

Lists the processes that have files open from the files specified in the command and provides other related information.

LISTOPENS supports SQL/MX ANSI names. FUP converts each ANSI name to the corresponding list of Guardian file names, then performs LISTOPENS on each of these files, and displays information of the opened files only.

Note. FUP support for fully qualified ANSI names for the LISTOPENS command is applicable on H06.04 and subsequent RVUs.

```
LISTOPENS [ / OUT listfile / ] fileset-list / ansiname-list
          [ , SCRATCH scratch-filename ]
```

OUT *listfile*

names a file or device to receive the listing output of the LISTOPENS command. You can use either a Guardian file name or a spool DEFINE name as the OUT *listfile*. If *listfile* is an existing file, FUP appends output to that file.

Note. For more information, see Specifying Files on page 1-8.

fileset-list

is a list of files or devices for which opens are to be displayed—including Enscribe files and all types of SQL/MP or SQL/MX files (except SQL/MP shorthand views). You can use wild-card characters. You cannot specify *qualified-fileset* for *fileset-list*. The *fileset-list* parameter can include device names in any of these forms:

$*device*
$*device*.#*name*
[$*device*].#*number*

name

is a device name made up of alphabetic characters, wild-card characters (* or ?), or a combination of both.

number

is a temporary file name made up of numbers, wild-card characters (* or ?), or a combination of both. If $*device* is not specified, it defaults to the current volume.

ansiname-list

ansiname-list = ‘ansiname’ [, ‘ansiname’]...

identifies SQL/MX ANSI Name catalogs, schemas, tables, indexes, partitions of tables and indexes, and any combination of these objects. A single quote (') is required to precede and delimit each ansiname. The ANSI names syntax is in accordance with Unified Syntax Proposal. The syntax is:

Note. For more description about *ansiname-list,* see ansiname-list on page 2-117.

SCRATCH *scratch-filename*

names a file or volume to be used for temporary storage during the sorting phase. If you omit this option, LISTOPENS uses a temporary file on the default volume.

LISTOPENS Listing Format

Example 2-11 shows the format the FUP LISTOPENS command uses to display file information. The first line of the display lists the file name specified in the LISTOPENS command.

Example 2-11. FUP LISTOPENS Listing Format

```
[ \node.]$volume.subvolume.file-id
  PID       MODE     USERID  SD  MYTERM      PROGRAM FILE NAME
  s,c,p     -pa -e   g,u     sd  term        prog-name
            -b
```

s

is the network node number of the node running the process that has the specified file open in the command.

c,p

is the processor number and process number of the process.

-p or *-b*

indicates that this is the primary or backup process (respectively) of a process pair.

a

is the access mode:

R	Read
R/W	Read and Write
E	Execute

-e

is the exclusion mode:

S	Shared
E	Exclusive
P	Protected

g,u

is the group ID, user ID of the process accessor ID.

sd

is the sync or receive depth specified by the process when the file was opened.

term

is the name of the home terminal of the process:

[\node.]$term

prog-name

is the program file name of the program that has the specified file open as it appears for a user process:

$volume.subvolume.file-id

It appears for a system process as:

$SYSTEM.SYS*nn*.OSIMAGE

$SYSTEM.SYS*nn* is the subvolume containing the operating system image that is currently in use (and *nn* is a two-digit octal integer.)

Note. For more information on automatic communication-path error recovery for disk files, see the *Guardian Programmer's Guide*. For a complete syntax description of the FILE_OPEN_ and OPEN procedures, see the *Guardian Procedure Calls Reference Manual*.

An example of how the LISTOPENS listing appears with data is:

```
-listopens mysvol.*
$GUEST.MYSVOL.EFILE
    PID        MODE  USERID  SD  MYTERM             PROGRAM FILE
NAME
215,01,0079    R/W-S 001,249 00  \FOXII.$LAM1.#ZWN
$SYSTEM.SYSTEM.TESTPROC
$GUEST.MYSVOL.RFILE
    PID        MODE  USERID  SD  MYTERM             PROGRAM FILE
NAME
215,01,0079    R/W-S 001,249 00  \FOXII.$LAM1.#ZWN
$SYSTEM.SYSTEM.TESTPROC
$GUEST.MYSVOL.SAMPLE
    PID        MODE  USERID  SD  MYTERM             PROGRAM FILE
NAME
215,01,0039    R/W-E 001,249 01  \FOXII.$TC1.#C13
$GUEST.FUPD00.FUP
```

LISTOPENS Guidelines

- To get information on temporary files, you must explicitly specify temporary file names using the # character.
- When you apply LISTOPENS to an SQL/MP protection view, it displays the processes that opened the view and the processes that opened the table view depends on.
- You cannot apply LISTOPENS to an SQL/MP shorthand view. If you include a shorthand view in *fileset-list*, LISTOPENS skips that view and issues a warning message.
- An open SQL base table might not be reported by LISTOPENS as open even if a FUP INFO command reports it is open. This situation occurs if the SQL base table is opened indirectly by a protection view. You can issue a FUP LISTOPENS

against all protection views on the SQL base table of interest. To list the protection views for a base table, use the command `SQLCI DISPLAY USE OF` *`tablename`*.

- The only device processes that currently give LISTOPENS information are disks, terminals, and X.25 lines.
- Fields that do not contain valid data for particular files in *`fileset-list`* are blank or zero filled.
- The information displayed in the USERID, MODE, and SD field for X.25 lines is:
  - The X.25 process does not use (or keep) the accessor ID, causing it to always return 000,000 for the accessor ID. This information appears in the USERID field for any process that has an X.25 data communication line open.
  - X.25 lines are always opened for read and write access. The display always shows R/W in the MODE field for any process that has an X.25 data communication line open.
  - An X.25 process does not use sync depth, causing the display to always show 0 in the SD field for any process with an X.25 data communication line open.
- If the device (controlling process) does not support a LISTOPENS request, FUP LISTOPENS displays:

  `WARNING - `*`dev name`*`: WILL NOT RETURN OPEN INFORMATION: ERR 2`
- LISTOPENS does not show a file as open if it was opened and closed during a transaction (although the transaction itself is still open). This situation occurs because the file has no openers. The INFO command does show as open files that have been opened and closed during a transaction (if the transaction itself is still open). This situation occurs because the file still has outstanding locks against it.
- LISTOPENS displays the message `Nonexistent process` if the process that is opening a file stops during the execution of the LISTOPENS command.
- The user running FUP must have remote access to any system, which is implicitly referenced by the ANSI name used in the FUP command. For example, if the ANSI name is 'CATALOG*', the user must have access to any node on which any visible catalogs reside, and so on. If the ANSI name is 'TABLE C.S.T', the user must have access to any node on which partitions of table C.S.T reside, and so on. Users who want to limit the scope of the command to all SQL/MX objects on the local machine can use a Guardian wildcard of the form: $*.ZSD*.*.
- The above explanation can be used to explain both an error 8551 from ANSI names or error 48 from the file system, depending on the command that was used.
- LISTOPENS supports SQL/MX objects, CATALOG, SCHEMA, TABLE, INDEX, and PARTITIONS.

LISTOPENS Examples

- To display a list of all processes that currently have open the file MYFILE (a file in the current default volume and subvolume):

  ```
  -LISTOPENS MYFILE
  ```

- To list all the opens of terminal $TERM:

  ```
  5>FUP LISTOPENS $TERM
  ```

- To list all the opens of X.25 device $DEVICE.#NAME:

  ```
  6>FUP LISTOPENS $DEVICE.#NAME
  ```

- To list all temporary files on the current volume beginning with 4:

  ```
  -LISTOPENS #4*
  ```

- To display a list of all processes that currently have opened the SQL/MX objects:

  ```
  FUP LISTOPENS 'SCHEMA CAT_ANSINAME01.SCH_ANSINAME01'
  $DATA05.ZSDHKPKT.CGJC4500
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  PID    MODE  USERID  SD  MYTERM             PROGRAM FILE NAME
  042,02,0432   R  -S  255,255 15  $ZTNT.#PTUJKRM
  $SYSTEM.SYSTEM.MXCI

  $DATA05.ZSDHKPKT.P1G73500
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  PID    MODE  USERID  SD  MYTERM             PROGRAM FILE NAME
  042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
  $SYSTEM.SYSTEM.MXCI

  $DATA1.ZSDHKPKT.J1C73500
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  PID    MODE  USERID  SD  MYTERM             PROGRAM FILE NAME
  042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
  $SYSTEM.SYSTEM.MXCI

  $DATA1.ZSDHKPKT.PJ183500
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  PID    MODE  USERID  SD  MYTERM             PROGRAM FILE NAME
  042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
  $SYSTEM.SYSTEM.MXCI

  $DATA2.ZSDHKPKT.KD973500
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  PID        MODE  USERID  SD  MYTERM         PROGRAM FILE NAME
  042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
  $SYSTEM.SYSTEM.MXCI

  $DATA2.ZSDHKPKT.Q6KB4500
  ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
  PID        MODE  USERID  SD  MYTERM         PROGRAM FILE NAME
  ```

```
042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
$SYSTEM.SYSTEM.MXCI

$DATA3.ZSDHKPKT.L9G93500
ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
PID        MODE  USERID  SD  MYTERM         PROGRAM FILE NAME
042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
$SYSTEM.SYSTEM.MXCI

$DATA3.ZSDHKPKT.SVS83500
ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
PID        MODE  USERID  SD  MYTERM         PROGRAM FILE NAME
042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
$SYSTEM.SYSTEM.MXCI

$DATA4.ZSDHKPKT.NQW83500
ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
PID        MODE  USERID  SD  MYTERM         PROGRAM FILE NAME
042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
$SYSTEM.SYSTEM.MXCI

$DATA4.ZSDHKPKT.WTK93500
ANSI NAME CAT_ANSINAME01.SCH_ANSINAME01.TAB1
PID        MODE  USERID  SD  MYTERM         PROGRAM FILE NAME
042,02,0432   R  -S 255,255 15  $ZTNT.#PTUJKRM
$SYSTEM.SYSTEM.MXCI
```

LOAD

Loads data into a structured disk file without affecting any associated alternate-key files. Data in the file being loaded is overwritten. This command applies only to Enscribe files.

To load any alternate-key files, use the command LOADALTFILE on page 2-136 after you complete a LOAD command.

```
LOAD in-filename , destination-filename
   [ , load-option ] ...


load-option is:

   EMPTYOK
   FIRST { ordinal-record-num                }
         { KEY { record-spec | key-value  } }
         { key-specifier ALTKEY key-value   }
   PAD [ pad-character ]
   in-option
   key-seq-option


   in-option is:

      BLOCKIN in-block-length
      [ NO ] COMPACT
      EBCDICIN
      RECIN in-record-length
      REELS num-reels
      [ NO ] REWINDIN
      SHARE
      SKIPIN num-eofs
      TRIM [trim-character ]
      [ NO ] UNLOADIN
      VARIN
      XLATE [ translation-table-name ]
      XLATEIN [ translation-table-name ]
      XLATEOUT [ translation-table-name ]


   key-seq-option is:

      MAX num-records
      PARTOF $volume
      SCRATCH scratch-filename
      SORTED
      DSLACK percentage
      ISLACK percentage
      SLACK percentage
```

You need to understand when to use the COPY, DUP[LICATE], and LOAD commands:

COPY	To change file attributes or copy files to or from nondisk devices
DUP[LICATE]	To create identical copies of disk files
LOAD	To create a structured disk file from scratch (much faster than COPY)

in-filename

names the file containing the records to be loaded. This file can be a disk file, a nondisk device, a process, a tape DEFINE name, an EDIT file, or a SPOOLER (code 129) file. You cannot use wild-card characters in *in-filename* or specify *qualified-fileset*.

destination-filename

specifies an existing disk file in which the records from *in-filename* are to be loaded. You cannot use wild-card characters in *destination-filename* or specify *qualified-fileset* for it. Any data already in *destination-filename* is overwritten by the LOAD process.

EMPTYOK

accepts an empty file for *in-filename*. If the IN file is empty (and you do not include the EMPTYOK option), the LOAD command terminates, and this message appears:

```
ERROR - EMPTY SOURCE FILE
```

If you include EMPTYOK (and LOAD encounters an empty IN file), this message appears:

```
RECORDS LOADED: 0
```

```
FIRST { ordinal-record-num                }
      { KEY { record-spec | key-value } }
      { key-specifier ALTKEY key-value  }
```

names the starting record of the input file for the copy. If you omit FIRST, the copy starts with the first record of the input file.

ordinal-record-num

is the number of records (from the beginning of the file) that are to be skipped. The first record in a file is record zero. If you specify this option for an unstructured disk file, the copy begins at:

*ordinal-record-num * in-record-length*

Note. The actual reading begins with the first record in the source file.

KEY { *record-spec* | *key-value* }

specifies the primary-key value for the starting record of a disk file. FUP begins reading the input file at the record you name with KEY.

Specify *record-spec* as an integer in the range 0 through (512000000 * 2,048) -1.

- Give the starting relative byte address for *record-spec* (for unstructured files).
- Give the starting record number for *record-spec* (for relative files).
- Give *ordinal-record-num* for *record-spec* (for entry-sequenced files).

Use *key-value* to indicate the approximate position of the starting record for key-sequenced files. Specify *key-value* as either *string* or:

```
"[" { string } [ , string ] ... "]"
    { 0:255  } [ , 0:255  ]
```

You can specify a list of strings with each string enclosed by quotation marks or integers representing byte values in the range 0 through 255. You must enclose the list of strings and integers (if specified) in square brackets.

Note. The brackets in *key-value* syntax are enclosed in quotation marks to indicate that they are part of the parameter—and not to indicate that the parameter is optional. Do not include the quotation marks when you type the brackets.

For example, specify a key value as the ASCII string “T905”, followed by a word containing the integer value zero, and a word containing the integer value nine:

```
[ "T905", 0, 0, 0, 9 ]
```

key-specifier

is a one-character or two-character string (specified inside quotation marks) specifying the alternate key to be used for positioning purposes.

ALTKEY *key-value*

specifies the alternate key of the starting record for disk files. FUP begins reading the input file at the specified record. Specify *key-value* for key-sequenced files according to the description of *key-value* in KEY { record-spec | key-value }.

PAD [*pad-character*]

specifies that records containing fewer than *in-record-length* bytes are padded with *pad-character* up to the record length specified in the file label. Specify *pad-character* as a single ASCII character inside quotation marks:

"*c*"

or as an integer in the range 0 through 255, specifying a byte value:

{ 0:255 }

Note. The *pad-character* default is an ASCII null character (binary 0).

in-option

specifies the format and control of *in-filename*. The value of *in-option* is any one of:

```
BLOCKIN in-block-length
[ NO ] COMPACT
EBCDICIN
RECIN in-record-length
REELS num-reels
[ NO ] REWINDIN
SHARE
SKIPIN num-eofs
TRIM [ trim-character ]
[ NO ] UNLOADIN
VARIN
XLATE [ translation-table-name ]
XLATEIN [ translation-table-name ]
XLATEOUT [ translation-table-name ]
```

Note. For a complete description of these options, see in-option on page 2-39.

key-seq-option

is an option for specifying the loading of key-sequenced files.

MAX *num-records*

is the number of records in *in-filename*. Specify *num-records* as a whole number in the range 0 through (512000000 * 2048) -1. This value does not need to be exact, but it should be equal to or greater than the actual number of records in *in-filename*. FUP uses *num-records* to determine the size of the scratch file used by the SORT process. The default *num-records* is 10,000.

If you specify SORTED, you can ignore this option.

PARTOF $*volume*

loads only the partition named in *destination-filename*. This option is for key-sequenced, partitioned files only. The $*volume* is the volume containing the primary partition of the destination file.

Note. For more information about the PARTOF option, see LOAD Guidelines on page 2-133.

SCRATCH *scratch-filename*

names a file or volume to be used for temporary storage during the sorting phase. If you omit this option, FUP uses a scratch file on the default volume.

If you specify SORTED, you can ignore this option.

SORTED

specifies that the records in *in-filename* are in the key-field order of the destination file, causing FUP to not sort the *in-filename* records. This option is for key-sequenced destination files only.

If you omit this option, FUP sorts *in-filename* records before loading *destination-filename*.

These three options specify the minimum percentage of space required by index blocks and in data blocks for future insertions. If space is not available when an insertion is made, a block split occurs.

DSLACK *percentage*

sets the minimum percentage of slack space in data blocks. Specify *percentage* as a value in the range 0 through 99. If you omit this option, FUP uses the SLACK percentage value.

ISLACK *percentage*

sets the minimum percentage of slack space for index blocks. Specify *percentage* as a value in the range 0 through 99. If you omit this option, FUP uses the SLACK percentage value.

SLACK *percentage*

sets the minimum percentage of slack space in both index and data blocks. Specify *percentage* as a value in the range 0 through 99. If you omit this option, FUP does not provide slack space (SLACK 0).

LOAD Guidelines

- The input records of key-sequenced files can be in sorted or unsorted order. If you do not specify SORTED, FUP invokes a SORT process to sort the records before loading the destination file.

- The FUP LOAD command reads the source file directly using large buffers unless you specify SHARE, in which case the Enscribe file system performs the reads.
- You can use a SORT DEFINE with the LOAD command. You must define it before starting FUP. For more information, see the *FastSort Manual*.
- For key-sequenced files, you can specify the percentage of slack space you want to keep for future insertions.
- If you specify the PARTOF option when loading a file that is not partitioned, you receive one of these error messages:
  - File-system error 11 (file not in directory) if the primary partition specified with PARTOF does not exist
  - The message BAD PARTITION PARAMETERS if the primary partition specified with PARTOF is not a partitioned file—or is not a partition of the file being loaded

Note. For more information about error messages, see Section 3, FUP Messages.

- Before you load partitioned files, consider these restrictions:
  - The range of keys for the different partitions is stored in the primary partition of a partitioned file.
  - If you try to load a secondary partition but you did not specify the PARTOF option, you receive an error message.
  - The FUP process loads all partitions if you do not specify PARTOF, and *destination-filename* is the primary partition.
  - To load a secondary partition, specify the name of the secondary partition as *destination-filename,* and specify the name of the volume where the primary partition resides (for the PARTOF option).
  - To load only the primary partition, specify the name of the primary partition as *destination-filename*, and specify the name of the primary volume for the PARTOF option.
- To sort the *in-filename* records, you must have both disk space for the sort scratch file and for *destination-filename* during the sorting phase.
- Use the PAD and TRIM options carefully in a FUP LOAD or COPY operation.

△ **Caution.** If your data contains *trim-character* or *pad-character*, data might be altered or lost.

An example of this situation occurs if you pad each record in a data file with zeros to a standard size in bytes and then store the records in another file. If you trim the trailing zeros when you execute a FUP LOAD or COPY of the stored records, any original data that ends with a zero is trimmed. To avoid this problem, use a value for *pad-character* or *trim-character* that is not contained in your data.

- The NO COMPACT option affects only relative files. If you include NO COMPACT in a LOAD command to load data from a nonrelative file, this message appears:

  ```
  WARNING - COMPACT OPTION IGNORED FOR NONRELATIVE FILES
  ```

- When you select the COMPACT option, and the source is a relative file that contains empty records, this message appears:

  source file : EMPTY RECORD FOUND AND NOT TRANSFERRED

  This message indicates that the target file has fewer records than the source file. It is issued only once, when the first empty record is encountered.

- If the input file for the FUP LOAD operation is a relative file that contains zero-length (empty) records (and you did not specify the NO COMPACT option), FUP ignores the zero-length records.

  This compacts records that are not zero length at the beginning of the file, and the records lose their relativity to position and key value. To avoid this problem, always include the NO COMPACT option to preserve the record position in relative files.

- When alternate-key records are not built because the full-alternate key does not exist within the primary record, this message appears:

  nnn RECORDS CONTAIN INCOMPLETE ALTERNATE KEY FIELDS (ALTERNATE KEY RECORDS NOT GENERATED)

- LOAD cannot load SQL files. You must use SQLCI LOAD instead.

- You cannot use the FUP LOAD command on queue files. Using the LOAD command can cause significant problems. FUP returns an error stating that this action on the queue file is not permitted.

LOAD Examples

- To load data from a relative file (RELFILE) to DFILE, causing zero-length records and data records to be transferred:

  ```
  -LOAD RELFILE, DFILE, NO COMPACT
  ```

- This example loads data from the tape device ($TAPE) into the key-sequenced disk file (KSFILE). Any records to be loaded from $TAPE are already sorted in the order of the key fields of KSFILE. The minimum percentage of slack space to remain in data blocks is 10 percent:

  ```
  -LOAD $TAPE, KSFILE, SORTED, DSLACK 10
  ```

- To load records from a file (OLDMAST), translate them using a translation table (MY_ENCRYPT), and write them to another file (SAVEMAST):

  ```
  -LOAD oldmast,savemast, XLATE my_encrypt
  ```

Commands Related to LOAD

COMMAND	Function	Page
DUP[LICATE]	Creates identical copies of disk files	2-61
COPY	Creates a record-by-record copy of a file	2-35
LOADALTFILE	Creates an alternate-key file from a primary file	2-136
BUILDKEYRECORDS	Creates alternate-key file records	2-21
RELOAD	Reorganizes an existing file online	2-146
CONFIG[URE]	Sets default options for the LOAD command	2-26

LOADALTFILE

Generates (from a primary file) the alternate-key records for a designated alternate-key file and then loads the records into the file. You can also specify the amount of slack space reserved for future insertions. This command applies only to Enscribe files. The LOAD command does not work on alternate-key files.

```
LOADALTFILE key-file-number , primary-filename
      [ , key-seq-option ] ...

key-seq-option is:

   MAX num-records
   SCRATCH scratch-filename
   DSLACK percentage
   ISLACK percentage
   SLACK percentage
```

key-file-number

selects the alternate-key file to load. Specify *key-file-number* as an integer in the range 0 through 255 to indicate an alternate-key file of the primary file. The alternate-key file must already exist. You can display the key-file number with the FUP INFO command on the primary file. This number might be different from the one used if you created the primary file with FUP.

primary-filename

names the existing primary file whose alternate-key records are to be generated and loaded into the file indicated by *key-file-number*. Partial file names are expanded using the current default node, volume, and subvolume names (if necessary).

key-seq-option

specifies options for loading the alternate-key file.

MAX *num-records*

specifies the number of records to be read from *primary-filename*. The default is 10,000,000. Specify *num-records* as a whole number in the range 0 through (512000000 * 2048) - 1. To determine the size of the scratch file that is used by the SORT process, FUP multiplies *num-records* by the number of alternate keys associated with *key-file-number*. The *num-records* value should be equal to or greater than the actual number of records in the primary file.

SCRATCH *scratch-filename*

specifies a file or volume to use for temporary storage while sorting. If you omit this option, FUP uses a scratch file on the default volume.

DSLACK *percentage*
ISLACK *percentage*
SLACK *percentage*

specify the minimum percentage of space to reserve in the index and data blocks for future insertions. If space is not available when an insertion is made, a block split occurs.

Note. For a complete description of these options, see LOAD on page 2-129.

LOADALTFILE Guidelines

- To perform a LOADALTFILE operation, you must have read and write access to the alternate-key files and read access to the primary file.
- LOADALTFILE ignores any NO UPDATE specifications.

  LOADALTFILE honors any NULL specification defined for a key field, but an alternate-key record is not generated for a field consisting entirely of null characters (if such a null character was defined).
- A sort operation is performed when you execute LOADALTFILE. The primary file is read sequentially according to its primary-key field. The alternate-key records are generated and written to the SORT process for each record read from the primary file. The sorted records are read from the SORT process and loaded into the indicated alternate-key file after the sort is complete. Disk space for the sort scratch file must exist during the sorting phase.
- You can use a SORT DEFINE with the LOADALTFILE command. You must define it before you start FUP. For more information, see the *FastSort Manual*.
- If the attributes of the alternate-key file are incorrect (for example, insufficient record length, key offset not zero, or not a key-sequenced file), LOADALTFILE fails and displays:

  ```
  ERROR - ALT FILE IS INCOMPATIBLE WITH ALT KEYS
  ```

- A duplicate key in an alternate-key file with the UNIQUE attribute causes LOADALTFILE to fail and display file-system error 71 (duplicate record).
- LOADALTFILE does not always generate alternate-key file records or display explanatory messages if either of these statements apply:
  - The full length of the alternate-key field is not contained in a specific primary record.
  - A null value was specified for the key, and the field contains only the null value.
- LOADALTFILE cannot work with SQL files. You must use SQLCI LOAD.

LOADALTFILE Example

This command generates alternate-key records from the primary file (KSFILE) and loads the records into the alternate-key file with key-file-number 0. A minimum of 10 percent slack space remains in the data blocks of the alternate-key file:

```
-LOADALTFILE 0, KSFILE, DSLACK 10
```

Commands Related to LOADALTFILE

COMMAND	Function	Page
LOAD	Creates a structured file from scratch	2-129
BUILDKEYRECORDS	Creates alternate-key file records	2-21

OBEY

Reads commands from the specified file and executes them.

After FUP reads an EOF in the command file, it returns you to the FUP command prompt. If the command file causes a change in status (by executing VOLUME or CONFIGURE commands), the new status remains enabled after the completion of the OBEY command.

```
OBEY filename
```

filename

is the name of the file containing the commands that you want to execute. The default volume and subvolume for the file (and all files in the command file) are derived using the standard FUP rules for defaulting any files enabled during the execution of the command.

OBEY Guidelines

- Command files must contain ASCII text with valid FUP commands. Command files are usually EDIT files but can be any other file type that FUP reads.

- Command file processing terminates with EOF or a FUP EXIT command.
- A command file can call other command files. A maximum of four command files can be active simultaneously.
- FUP displays the commands in a command file only if the CONFIGURE ECHO OBEY option is enabled. This option puts commands from a command file into the HISTORY buffer only if this option is enabled. By default, ECHO OBEY is enabled.
- Any errors encountered during the execution of a command file are listed at the home terminal (or list file, if applicable) and are handled as normal errors according to the ALLOW conditions currently enabled. For example, a severe error aborts all OBEY command-file processing (and the FUP session) unless ALLOW*num* SEVERE ERRORS was enabled when the error occurred and the error count had not been exceeded.

OBEY Example

To read commands from a specified file (ALLSUBS) and execute them:

```
1> FUP OBEY ALLSUBS
```

FUP is started using the OBEY command to execute FUP commands in the specified file (ALLSUBS). This example writes to the terminal because there is no OUT file. Control of the terminal returns to TACL after FUP executes the last command in the command file.

PURGE

Deletes a single disk file, a set of files, many sets of files, or an entire subvolume of files. This command applies only to Enscribe files.

```
PURGE [ ! ] fileset-list [ , [ NO ] LISTALL ] [ ! ]
```

!

in either or both of the displayed positions, indicates to purge the files without prompting for permission.

If the CONFIGURE NO PROMPT PURGE option is enabled, *!* is assumed implicitly, and prompting does not occur.

If you are not running FUP interactively but are entering FUP commands through an IN file or another process, the *!* is required. If you omit *!*, you get a syntax error, and FUP does not purge any files.

If you are running FUP interactively (entering FUP commands at the command interpreter or FUP prompt) and you omit *!*, FUP prompts you for permission to purge the *fileset-list*:

- If *fileset-list* is a single file (or a list of single files), the FUP INFO listing is displayed for each file and followed by the PURGE? prompt. Type Y or y to purge the file. If you type any other response, the file is not purged.
- If *fileset-list* includes an entire volume or subvolume (or if you use the wild-card option to specify the subvolume or file ID), you are prompted for permission to purge each file set as it is encountered in *fileset-list*:

```
DO YOU WISH TO PURGE THE ENTIRE FILESET fileset ?
( Y[ES], N[ONE], S[ELECT], F[ILES] )?
```

  Type your answer from the four choices displayed and press RETURN:

  - If you type Y, y, or yes, the file set is purged with no more prompting.
  - If you type N, n, or none, the file set is skipped and PURGE continues with the remainder of the *fileset-list*.
  - If you type S, *s*, or select, the FUP INFO listing is displayed for each file in the file set, and it includes the PURGE? prompt. Type Y(es) or N(o) for each file to specify whether to purge it.
  - If you type F, f, or files, a FILES display appears for the file set, followed by a prompt for the entire file set with the same four choices.
  - If you press CTRL-Y, PURGE terminates, and the FUP prompt reappears.
  - If you press RETURN, PURGE operates under select.
  - If you enter any other response, FUP ignores your response and redisplays the file set prompt.

Note. For different responses to the file-set prompt, see PURGE Examples on page 2-142.

fileset-list

names a file, a set of files, or many sets of files to purge. Partial file names are expanded using the current default node, volume, and subvolume. You can use wild-card characters and specify *qualified-fileset* for *fileset-list*.

If you are including the LISTALL or NO LISTALL option, you must place parentheses around *fileset-list*. If you omit the parentheses, FUP treats LISTALL or NO LISTALL as file names. After FUP executes PURGE, it displays the total number of files it purged.

[NO] LISTALL

specifies whether to list the names of all files as they are purged. The listing is:

$*volume.subvolume.file-id* PURGED.

LISTALL is the default in interactive mode. The default for noninteractive mode is NO LISTALL.

If you include this option, you must enclose *fileset-list* in parentheses even if the list contains only one file. If you omit the parentheses, FUP treats LISTALL or NO LISTALL as file names.

If you omit the parentheses and specify NO LISTALL, FUP returns a `required delimiter is missing` error message. If you omit the parentheses and specify LISTALL, FUP searches for a LISTALL file in the current default subvolume. If no file with that name exists, it returns file-system error 11 (file not in directory).

`PARTONLY`

purges only the specified partition for a partitioned file.

If you include this option, you must enclose *fileset-list* in parentheses even if the list contains only one file. If you omit the parentheses, FUP treats PARTONLY as a file name.

This option is not valid for any Enscribe object. If you use PARTONLY for Enscribe objects, PURGE terminates abnormally (ABENDs) with an error.

PURGE Guidelines

- If you try to purge a file that has transaction-mode record or file locks pending and it is audited by TMF, the purge request fails with file-system error 12 (file in use). Even if the processes that opened the file no longer exists, the file-system error still occurs.
- PURGE cannot dispose of SQL files that are not SQL object files. Instead, you must use SQLCI PURGE.
- For objects compiled by SQL, PURGE displays error 197 (an SQL error has occurred).
- You can purge a file only if it is not in use (that is, it cannot be open, running, or undergoing a backup process). You also must have purge access or be logged on as the super ID (255,255).
- If you try to purge a file before its expiration date (NOPURGEUNTIL attribute), an error occurs. To alter the expiration date, use the ALTER command.
- If you purge a file that has the CLEARONPURGE option set (for more information, see SECURE on page 2-161), the disk process physically deletes the file data from the disk (overwrites it with blank data) and then deletes the file name from the directory.
- In the LISTALL mode, PURGE lists each file name immediately after deleting it. If you press the BREAK key while FUP is running, the PURGE command terminates and does not delete any remaining files in *fileset-list*.

PURGE Examples

- To purge the file WKLYRPRT in the current default subvolume without prompting for permission:

```
-PURGE WKLYRPRT !
$MYVOL.RECDS.WKLYRPRT PURGED.
1 FILE PURGED
```

- To purge an entire file set without prompting for permission by including the *!* option:

```
 -PURGE $VOL1.SVOL.*FILE !
 $VOL1.SVOL.MYFILE PURGED.
 $VOL1.SVOL.NEWFILE PURGED.
 $VOL1.SVOL.OLDFILE PURGED.
 3 FILES PURGED
 -
```

- To purge an entire file set without being prompted (and without getting a list of the files purged), include the *!* and NO LISTALL options:

```
 -PURGE ($VOL1.SVOL.*), NO LISTALL !
 3 FILES PURGED
 -
```

- To purge all files other than EDIT files that start with the letter M or later:

```
PURGE * WHERE NOT FILECODE=101 START M
```

- To display the FUP INFO listing for each file in the specified file set and prompt for permission to purge each file:

```
-PURGE WKLYRPRT, MONTHEND, REQTOTAL
         CODE  EOF    LAST MODIF  OWNER RWEP  TYPE  REC
BLOCK
$MYVOL.RECDS
 WKLYRPRT     2048 29MAR87 11:46  8,44  AOAO   R     11
1024
PURGE? Y
$MYVOL.RECDS.WKLYRPRT PURGED.
         CODE  EOF    LAST MODIF  OWNER RWEP  TYPE  REC
BLOCK
$MYVOL.RECDS
 MONTHEND     5120 29MAR87 11:44  8,44  AOAO   R     10
1024
PURGE? N
         CODE  EOF    LAST MODIF  OWNER RWEP  TYPE  REC
BLOCK
$MYVOL.RECDS
 REQTOTAL     8192  2MAR87 15:38  8,44  CUCU   K     11
1024
PURGE? Y
$MYVOL.RECDS.REQTOTAL PURGED.
2 FILES PURGED
  -
```

- To purge a file set and be prompted for permission (for the entire file set and for individual files), omit the *!* option and specify the subvolume to purge.

  These examples show what happens when you choose each of the four possible responses to the file set prompt. Each example starts with this command:

```
-PURGE $A.B.FILE*
DO YOU WISH TO PURGE THE ENTIRE FILESET $A.B ?
( Y[ES], N[ONE], S[ELECT], F[ILES] )?
```

  If you type Y, the entire file set is purged:

```
( Y[ES], N[ONE], S[ELECT], F[ILES] )? y
$A.B.FILE1 PURGED.
$A.B.FILE2 PURGED.
$A.B.FILE3 PURGED.
3 FILES PURGED
```

  If you type N, no files are purged:

```
( Y[ES], N[ONE], S[ELECT], F[ILES] )? N
0 FILES PURGED
```

If you type S (or press Return), the FUP INFO listing appears for each file in the file set, followed by the PURGE? prompt:

```
( Y[ES], N[ONE], S[ELECT], F[ILES] )? S
          CODE  EOF    LAST MODIF  OWNER RWEP  TYPE...
$A.B
 FILE1     101  1646  5APR85 15:55  8,44  AO--
PURGE? Y
$A.B.FILE1 PURGED.
 FILE2     101  1646  4APR85 16:31  8,44  CUCU
PURGE? N
 FILE3     101  1646  3APR85 12:48  8,44  CUCU
PURGE? Y
$A.B.FILE3 PURGED.
2 FILES PURGED
```

If you type F, the contents of the file set are listed, and the original prompt appears again:

```
( Y[ES], N[ONE], S[ELECT], F[ILES] )? f
$A.B
    FILE1     FILE2     FILE3
DO YOU WISH TO PURGE THE ENTIRE FILESET $A.B ?
( Y[ES], N[ONE], S[ELECT], F[ILES] )?
```

Commands Related to PURGE

COMMAND	Function	Page
CONFIG[URE]	Can override the default *!* option	2-26
PURGEDATA	Purges the data from a file	2-144

PURGEDATA

Removes all data from a file. This command applies only to Enscribe files.

PURGEDATA does not physically purge data. It purges data logically by setting the end-of-file (EOF) pointer to zero—the relative position to the beginning of the file. The file still exists, its file name can be displayed (with the FILES command), and the extents remain allocated (until you issue a DEALLOCATE command for the file).

```
PURGEDATA fileset-list [ , PARTONLY ]
```

fileset-list

is a list of files from which data is to be purged. Partial file names are expanded using the current default node, volume, and subvolume (if necessary). You can use wild-card characters and can specify *qualified-fileset* for *fileset-list*.

PARTONLY

purges data in any primary or secondary extents of partitioned files that reside in *fileset-list*. If you omit PARTONLY, data is purged from all partitions of partitioned files—but only if the primary partitions of the files reside in *fileset-list*.

PURGEDATA Guidelines

- To use PURGEDATA and purge data from a file, you must have write access to that file or be a super-group user (255, *n*).
- If you try to purge data from an audited file, TMF must be running, and the disk containing the file must be enabled for TMF.
- For a PURGEDATA operation, the FUP process tries to open the specified file with exclusive access. If the file cannot be opened with exclusive access, the PURGEDATA command fails; for example, if the file is already open.
- The CLEARONPURGE option set with the FUP SECURE command has no effect on PURGEDATA. After the PURGEDATA command is executed, the data is physically present on the disk (but inaccessible) until it is overwritten by new data or deallocated with the DEALLOCATE command.
- PURGEDATA cannot purge SQL files that are not SQL object files. Instead, you must use SQLCI PURGEDATA.

PURGEDATA Example

To logically purge the data from MYFILE in the current default subvolume:

```
-PURGEDATA MYFILE
```

Commands Related to PURGEDATA

COMMAND	Function	Page
PURGE	Deletes a file	2-139
DEALLOCATE	Deallocates unused file extents	2-60

RELOAD

Physically reorganizes a key-sequenced file or SQL object (table or index only) while allowing shared read and write access to the file or object.

FUP RELOAD supports key-sequenced file reorganization of volume directories, TMF-audited files and SQL tables, and nonaudited files and SQL tables. A reload operation improves access time and use of space for a key-sequenced file or SQL object that has undergone many insertions, deletions, and updates with length changes.

FUP RELOAD supports SQL/MX ANSI names. FUP converts each ANSI name to the corresponding list of Guardian file name and then performs RELOAD on these files.

Note. FUP support for fully qualified ANSI names for the RELOAD command is applicable on H06.04 and subsequent RVUs.

```
RELOAD [ / OUT listfile / ] filename / 'ansiname'
       [ [ NO ] DEALLOCATE   ]
       [ , NEW               ]
       [ , PARTOF $volume    ]
       [ , RATE percentage   ]
       [ , DSLACK percentage ]
       [ , ISLACK percentage ]
       [ , SHARE             ]
       [ , SLACK percentage  ]
       [   RECLAIM           ]
```

`OUT listfile`

names a file or device to receive the output of the RELOAD command. You can use either a standard file name or a spool DEFINE name as the OUT *listfile*. If *listfile* is an existing file, FUP appends output to that file.

Note. For more information about *listfile*, see Specifying Files on page 1-8.

---

filename

is the name of the key-sequenced file or SQL object (table or index only) to be reorganized. It can also be a secondary partition of a partitioned file. For more information, see PARTOF $volume on page 2-148. You cannot use wild-card characters or specify *qualified-fileset* for *filename*.

ansiname

identifies SQL/MX ANSI name table partition or an index partition. A single quote (') is required to precede and delimit *ansiname*.

ansiname ::= {TABLE | INDEX} *SQL-name* PARTITION *SQL-identifier*

SQL-name

is used to name base SQL base objects (such as tables or indexes) in addition to their SQL containers: catalogs and schemas. The names (called 3-part names) for SQL base objects such as tables, indexes, or modules are composed of three SQL identifiers separated by two dot characters (for example, CAT.SCH.T).

SQL-identifier

is a name used by SQL/MX to identify tables, views, columns, and other SQL entities. SQL identifiers can be either regular or delimited and can contain up to 258 characters in external form, or equivalently up to 128 characters in internal format. Regular identifiers begin with a letter (A through Z or a through z), but can also contain digits (0 through 9), or underscore characters (_).

Regular identifiers used to name a SQL/MX module (the basic object part) can start with the ^ character or contain the ^ character.

Note. The information regarding SQL/MX module provided above is for reference purpose only. FUP commands do not support the MODULE keyword.

A delimited identifier is enclosed in double quotes ("). Delimited identifiers are character strings that appear within double quote characters (") and consist of alphanumeric characters and other characters, except for character @, /,\, and ^. To include a double quote character in a delimited identifier, use two consecutive double quotes. A delimited module name in SQL/MX can contain the circumflex character (^).

`[ NO ] DEALLOCATE`

the default (DEALLOCATE) lets you compact a file and delete all space beyond the new EOF.

If you select NO DEALLOCATE, the FUP process does not deallocate any extent beyond the new EOF that is set by RELOAD.

`NEW`

specifies to perform a new reload operation on *filename* even if there is a suspended reload for *filename*. If NEW is not specified (and a previous reload for *filename* was suspended or stopped), FUP restarts the previous operation from the point where it stopped.

NEW does not override a reload operation in progress. If NEW is specified and a reload for *filename* is in progress, FUP displays an error message.

PARTOF $*volume*

specifies the volume where the primary partition resides if *filename* is an Enscribe secondary partition. If you specify a secondary partition but do not specify the PARTOF option, the system returns an error message.

The PARTOF option is not required for an SQL object. If you specify this option for an SQL object, FUP ignores it.

RATE *percentage*

specifies that the reload operation should spend *percentage* of its time executing the reorganization and the rest of its time delaying. Specify *percentage* as an integer in the range 1 through 100. The default value is 100.

If you are restarting a reload operation and do not specify a new rate value, FUP uses the value from the previous reload.

DSLACK *percentage*

sets the minimum percentage of slack space in data blocks. Specify *percentage* as a value in the range 0 through 99. If you omit this option, FUP uses the SLACK percentage value.

ISLACK *percentage*

sets the minimum percentage of slack space in index blocks. Specify *percentage* as a value in the range 0 through 99. If you omit this option, FUP uses the SLACK percentage value.

SLACK *percentage*

sets the minimum percentage of slack space in both index and data blocks. Specify *percentage* as a value in the range 0 through 99. If you omit the SLACK parameter (and the DSLACK and ISLACK parameters), FUP uses a default value of zero for Enscribe files and 15 for SQL objects. If you are restarting a reload that has been suspended, FUP uses the slack values from the previous reload operation.

The SLACK value overrides any DSLACK and ISLACK values. If you specify an ISLACK or DSLACK value (and then specify a SLACK value), FUP uses the SLACK value and ignores the DSLACK or ISLACK value.

SHARE

allows a RELOAD operation on a file to occur concurrently with update operations being executed by other applications or allows a user to open a file after a RELOAD operation has been initiated if the user does not have write permission for the file.

By default, a RELOAD operation cannot be performed if the user initiating the operation does not have write access to a file whether the SHARE option is specified or not. If the user does not have write access, error 48 is returned.

To perform a RELOAD operation, a user who does not have write access to the file can use the SHARE option by configuring ORSERV (online reload server):

Note. These steps can be performed only on the ORSERV object, which has symbols information.

```
TACL>TAL ;suppress
*INT PROC CHECK^RELOAD^FLAG; BEGIN RETURN 1; END;
*|EOF
TACL>BIND
*ADD * FROM  orserv-path
*REPLACE * FROM OBJECT
*BUILD ORSERV!
*|EOF
TACL>PURGE OBJECT
```

If the newly built ORSERV object is used for a RELOAD operation, the SHARE command works even without write access.

If RELOAD is performed with the SHARE value, the default DEALLOCATE is ignored, and a warning is returned.

If RELOAD is performed with the SHARE value and suspended, it still has the SHARE attribute when it is restarted.

If RELOAD is performed with the SHARE value and the requisite DP2 is not present, error 49 is returned.

`RECLAIM`

reclaims the unclaimed free space for a SQL object caused by a Data Definition Language (DDL) move partition boundary or one-way split operation.

This option works only with the required DP2 option installed in the system. A suspended RELOAD with the RECLAIM option automatically takes the RECLAIM value when RELOAD is restarted.

The RECLAIM option is not valid for any Enscribe object. If RELOAD is performed with the RECLAIM value on any Enscribe object, error 2 appears.

This value is available only in G-series RVUs.

The free space is reclaimed on a subsequent RELOAD without the RECLAIM option.

RELOAD Guidelines

- RELOAD must be used with a C30 or later product version of TMF. Using RELOAD with earlier product versions of TMF might produce new information in the file that is unrecoverable.

- RELOAD does not require files to be audited.
- If reload a volume directory during a peak production period, run RELOAD at a low rate to reduce the degradation of system performance caused by the RELOAD.
- D-series software supports allocation (with the ALLOCATE command, for Enscribe only) and deallocation of volume directory extents, with the [NO] DEALLOCATE parameter.

  This deallocates unused extents when you issue the RELOAD command against a volume directory. If you want unused extents in a directory, use the ALLOCATE command after each reload.
- When you issue a RELOAD command, FUP creates an online reload server (ORSERV) process (on the node of the file) to perform the reload. If FUP cannot create the ORSERV process, an error message appears.
- The ORSERV process created by a RELOAD command runs in the NOWAIT state. While the reload operation is processing, FUP displays its hyphen (-) prompt so you can perform other FUP operations.
  - To monitor the status of a reload operation, use the STATUS command.
  - To temporarily stop the reload, use the SUSPEND command.
- Access to *filename* by an application (during the reload) requires shared access.
- The reload operation on an audited file generates audit records that describe the movement of data within the file. The audit record total from a file can triple the file length (discounting free space).
- A reload operation can degrade system performance. To control the processor time used by the reload, use the RATE parameter. Any percentage below 100 prevents the reload from monopolizing the processor and its resources.
- To prevent system performance degradation, do one of:
  - Increase the TMF audit-trail file configuration before you use FUP RELOAD.
  - Reduce the rate of audit-trail generation by decreasing the RATE percentage in the RELOAD command.
- If you are restarting a reload operation and do not specify new rate and slack values, FUP uses the values from the previous reload operation.
- To reload all of a key-partitioned file’s partitions, you must reload each partition separately. If the primary partition is the only one specified in the RELOAD command, it is the only partition that is reloaded.
- An online file reload can generate a large amount of audit-trail information (in a short amount of time) for TMF-audited files or tables. The amount of audit information generated increases if you concurrently run multiple reloads.

- The reload operation creates and maintains the file ZZRELOAD.ZZRELOAD, which contains status information for the current and past reload operations. There is one ZZRELOAD file per volume, created the first time a reload is run on it.
- The reload operation requires a small amount of disk space to reorganize the specified file—even if the result of the operation is a smaller file. A reload operation fails if insufficient disk space is available on the volume for RELOAD (ORSERV) to perform the file reorganization.
- When you run RELOAD on an SQL partitioned file concurrent to the split of another partition, it terminates abnormally (ABENDs) with error 60. FUP treats this error as a suspended operation and displays this message and the suspended information in the STATUS command:

  ```
  ERROR:RELOAD SUSPENDED DUE TO DDL OPERATION
  ```

- The user running FUP must have remote access to any system, which is implicitly referenced by the ANSI name used in the FUP command. For example, if the ANSI name is 'TABLE C.S.T', the user must have access to any node on which partitions of table C.S.T reside, and so on.
- The above explanation can be used to explain both an error 8551 from ANSI names or error 48 from the file system, depending on the command that was used.
- RELOAD supports SQL/MX objects, TABLE PARTITION, and INDEX PARTITION.

RELOAD Example

```
-RELOAD PAYFILE, NEW, RATE 40

-RELOAD 'TABLE CAT_ANSINAME01.SCH_ANSINAME01."TABLE5" PARTITION
(PART1)'
```

Commands Related to RELOAD

COMMAND	Function	Page
STATUS	Displays the status of a RELOAD operation	2-183
SUSPEND	Suspends a RELOAD operation	2-186
CONFIG[URE]	Sets default options for the RELOAD command	2-26

RELOCATE

Moves files on SMF virtual disks from one physical volume to another within a storage pool. This command is available only to the file owner or a super-group user (255,*n*).

```
RELOCATE logical-set  [ , physvol ] [ PRIORITY nnn ]
```

logical-set

specifies a set of logical files where relocation is to occur. You can use wild-card characters or specify a qualified file set. The *logical-set* can also include SQL files.

physvol

specifies the name of a physical volume.

nnn

specifies the priority of a job (1 through 199). Changing this option lets you run RELOCATE at a lower priority if you do not want it to interfere with other jobs.

RELOCATE Guidelines

- If multiple files are specified in the *logical-set*, FUP relocates them one at a time (waited).
- If a file in *logical-set* is not an SMF file or if *physvol* is not a member of the SMF storage pool that the virtual disk is associated with for each file in *logical-set*, it is an error.
- Errors encountered during this command are displayed by the normal FUP error-handling mechanisms and terminate the command or not according to the normal FUP ALLOW rules.
- If a file in *logical-set* is an SQL/MP base table with protection views, the protection views are also moved. The protection views do not have to be in the file-set list.
- If a file in the file-set list is an SQL/MP protection view, it is not moved unless the base table it is defined on is also moved.
- If the PHYSVOL option is not specified, SMF chooses the best volume on the virtual disks.

RELOCATE Example

To relocate the files starting with BLUE from $L.SMS to the physical volume $ABC:

```
-RELOCATE $L.SMS.BLUE*, $ABC
```

RENAME

Changes the file name or subvolume name of a disk file or renames sets of files. This command applies only to Enscribe files.

You can also use RENAME to reorganize your Enscribe files by renaming one or more files in one or more subvolumes to a single subvolume. If the destination subvolume

does not already exist, the renaming process creates it. To transfer files from one volume to another, you must use FUP DUP or FUP COPY—RENAME cannot do it.

```
RENAME old-fileset-list , new-fileset [ , PARTONLY ]
```

old-fileset-list

specifies the files to be renamed. You can use wild-card characters and can specify *qualified-fileset* for *fileset-list*.

For example, *SF finds all the files that end with SF. You can use ?SF to find all files that start with one character and are followed by SF. The files in *old-fileset-list* can reside on more than one subvolume.

new-fileset

specifies the new names of the files. You cannot specify *qualified-fileset* for *fileset*, and you must specify the disk file-name portion of *new-fileset* as an asterisk (*) to rename more than one file. If you do this, each file is given the disk-file name of its corresponding input file, and the new subvolume name is taken from *new-fileset*. The files in *new-fileset* must reside on one subvolume.

PARTONLY

specifies that only partitions included in *old-fileset-list* are to be renamed (for partitioned files). If you omit PARTONLY, FUP renames all the partitions for partitioned files if their primary partitions reside in *old-fileset-list*. PARTONLY has no effect on files that are not partitioned.

RENAME Guidelines

- If you try to rename a file whose AUDIT attribute is set for auditing by TMF, the rename request fails, and you receive file-system error 80 (invalid operation on audited file or nonaudited disk volume).
- You cannot rename a file that is currently open with exclusive access.
- To rename a file, you must have purge access to the file or be the super ID (255,255). To change a file's contents, you must have write access to the file.
- RENAME cannot work with SQL/MP or SQL/MX files that are not object files.
- If you do not own all the files in *old-fileset-list*, use the ALLOW option (from the CONFIGURE command) with SEVERE ERRORS set to a high number. This lets the RENAME command finish. The command can rename only files that you own.

RENAME Example

To change the subvolume name of all files in the current default subvolume (and all files in the SVOL subvolume) to NEWSVOL, having all files retain their original file names:

```
-RENAME (SVOL.*, *), NEWSVOL.*
```

REPORTWIDTH

Sets the maximum length (in columns) for FUP to format its output. This command changes the normal (default) 132-character output format of these FUP commands to a smaller output format:

- COPY with a DUMP option
- FILES
- LISTLOCKS
- SUBVOLS

The REPORTWIDTH command became an option of the CONFIG[URE] command with the D30 product version of FUP. However, for compatibility, FUP product versions starting at D30 continue to recognize the REPORTWIDTH option as command syntax. For more information, see CONFIG[URE] on page 2-26.

RESET

Restores one or more file-creation attributes to the default settings. For a list of file-creation attributes and their default values, see SET on page 2-165.

```
RESET [ reset-opts | CONFIG[URE] config-simple-opts ]

reset-opts is:

create-spec [ , create-spec ] ...

create-spec is:

ALTCREATE
ALTFILE [ key-file-number ]
ALTFILES
ALTKEY [ key-specifier ]
ALTKEYS
AUDIT
AUDITCOMPRESS
BLOCK
BUFFERED
CODE
COMPRESS
DCOMPRESS
EXT
FORMAT
ICOMPRESS
KEYLEN
KEYOFF
MAXEXTENTS
ODDUNSTR
PART [ partition-num ]
PARTONLY
PARTSREC
PHYSVOL
QUEUEFILE
REFRESH
SERIALWRITES
TYPEBUFFERSIZE
VERIFIEDWRITES
```

```
config-simple-opts
  config-simple-option [ , config-simple-option ]

config-simple-option is:

ALLOW
DISPLAYBITS
ECHO [CONFIG[URE] | OBEY
IOTIMEOUT
NETBLOCKSIZE
PROMPT [ PURGE ]
REPORTWIDTH
RESTARTUPDATE
STATONLY
XLATE [ xlate-table-name ]
XLATEIN [ xlate-table-name ]
XLATEOUT [ xlate-table-name ]
COPY [ copy-option ]
DUP[LICATE] [ dup-option ]
LOAD [ load-option ]
LOADALTFILE [ loadaltfile-option ]
RELOAD [ reload-option ]
```

RESET Guidelines

- Each *create-spec* keyword in a RESET command resets the corresponding creation attribute to its default setting. If you do not include *create-spec* in your RESET command, FUP resets all creation attributes to their default settings.
- If you specify any of the plural keywords (ALTKEYS, ALTFILES, or PARTS), FUP resets all the corresponding creation values (alternate keys, alternate-key files, or secondary partitions).

Note. For the *create-spec* parameter default values, see SET on page 2-165. For *config-simple-opts*, see CONFIG[URE] on page 2-26.

RESET Examples

- This example assumes you set the file-creation attributes to create a relative structured DP2 file $COMPUTR.BOOKS.PASCAL (which is not displayed in this example) with a record size of 10 bytes and an alternate-key file (named $COMPUTR.BOOKS.BLAISE). The SHOW command displays:

```
SET LIKe $COMPUTR.BOOKS.BLAISE
TYPE R
EXT ( 1 PAGES, 1 PAGES )
FORMAT 1
REC 10
BLOCK 1024
ALTKEY ("aa", FILE 0, KEYOFF 0, KEYLEN 5)
ALTFILE (0, $COMPUTR.BOOKS.BLAISE)
```

```
ALTCREATE
MAXEXTENTS 16
```

Reset the record size file-creation attribute to the default value of 80 bytes:

```
-RESET REC
```

After you complete the reset, the FUP SHOW command includes this line in its display:

```
REC 80
```

- This example uses a single RESET command with no *create-spec* to restore all the defaults. This example includes the MAXEXTENTS and BUFFERSIZE DP2 file attributes:

```
-RESET
-SHOW
   TYPE U
   EXT ( 1 PAGES, 1 PAGES )
   FORMAT1
   MAXEXTENTS 16
   BUFFERSIZE 4096
```

- To reset all the DUP[LICATE] options for the CONFIG[URE] command:

```
-RESET CONFIGURE DUP
```

Commands Related to RESET

COMMAND	Function	Page
SET	Sets default file attributes	2-165
SHOW	Displays default file attributes or CONFIG[URE] options	2-180
CREATE	Creates a file using default file attributes	2-57
CONFIG[URE]	Sets default options for FUP	2-26

RESTART

Restarts a RESTARTABLE DUP operation at the point where it failed. The operation continues from near the point where it failed.

```
RESTART [ restart-filename ]
```

restart-filename

is the name of an unstructured disk file created by a previous DUP operation with the RESTARTABLE option specified. The file contains information describing the progress of the operation.

If *restart-filename* is not specified, FUP searches your current subvolume for a file named ZZRSTART.

RESTART Guidelines

- FUP (DUP) updates the restart file periodically. This causes a RESTART operation to start from the last update, which is not necessarily from the last record written.
- The RESTART operation fails if any of these conditions exist:
  - The last modified time of the source file has changed.
  - A file with the same name as *restart-filename* exists on the destination disk for the DUP operation.
- FUP does not verify if the security, ownership, or other attributes of the source file have changed since the last modified time of the file. The file attributes appear in the file label of the source file.
- During a RESTART operation, FUP makes additional updates to the restart file to record its progress. If the RESTART operation fails, a subsequent RESTART operation can continue from the point where the failed operation stopped.
- The RESTART file is purged when the DUP[LICATE] process finishes.

RESTART Examples

- This example assumes that the DUP operation from the first command fails. The RESTART command restarts the DUP operation.

  ```
  -DUP FILE1, FILE2, RESTARTABLE
  -RESTART
  ```

  If a RESTARTABLE DUP operation fails and you issue a RESTART command, FUP always echoes the original DUP command in uppercase.

  A *restart-filename* is not included in the RESTARTABLE option of the DUP command, so FUP searches for the ZZRSTART file on the current subvolume for the information it needs to perform the RESTART operation.

- To include a *restart-filename* with the RESTARTABLE DUP command:

  ```
  -DUP OLDFILE, NEWFILE, RESTARTABLE RSFILE
            .
            .
  -RESTART RSFILE
  ```

  If the DUP operation fails, the RESTART command must specify the name of the RSFILE to request FUP to use the information contained in the unstructured file (file code 855) RSFILE for the RESTART operation.

Commands Related to RESTART

COMMAND	Function	Page
DUP[LICATE]	Creates an identical copy of a disk file	2-61

REVOKE (Super ID)

Revokes a license for a privileged program file, or resets the security attributes of files and programs that have standard security codes. To revoke attributes of files protected by the Safeguard product, use the Safeguard command interpreter (SAFECOM). This command applies only to Enscribe files.

REVOKE command privileges are determined by your user ID:

- If your user ID identifies you as the owner of a file, you can reset the CLEARONPURGE and PROGID security attributes of the file by including the file in *fileset-list* and specifying the attribute you want to reset.
- The super ID (255, 255) can revoke the license for a privileged program file named in *fileset-list* by omitting *secure-option* from the command.

```
REVOKE fileset-list [ , secure-option ] ...

secure-option is:

   CLEARONPURGE
   PARTONLY
   PROGID
```

fileset-list

is a list of files whose licenses or other attributes are to be revoked. Partial file names are expanded using the current default node, volume, and subvolume (if necessary). You can use wild-card characters and specify *qualified-fileset* for *fileset-list*.

secure-option

is one of three options that can be set by a program or by a SECURE command. For more information, see SECURE on page 2-161.

CLEARONPURGE

physically deletes all data within *fileset-list* from the disk (by overwriting the file space with blank data) when the file is purged.

Including CLEARONPURGE in a REVOKE command revokes the CLEARONPURGE option for the file. When you purge a file that does not have CLEARONPURGE set, the disk space is logically deallocated. This option has no effect on a PURGEDATA operation.

PARTONLY

specifies that only the designated partition is affected by this REVOKE command (for partitioned files). If you omit PARTONLY, every partition is

affected. If PARTONLY is the only *secure-option* in a REVOKE command, the license for that partition is revoked.

PROGID

sets the process accessor ID to the owner ID of the program file when the program file is run (for program files only). Including PROGID in a REVOKE command revokes the PROGID option for the file. The process accessor ID is set to the ID that corresponds to the creator of the process when the program is run.

REVOKE (Super ID) Guidelines

- To revoke the LICENSE attributes of a file, you cannot specify the CLEARONPURGE attribute and the PROGID attribute.
- To revoke files protected by Safeguard, you must use the Safeguard command interpreter (SAFECOM). If you issue the FUP REVOKE command for a Safeguard protected file, file-system error 199 (disk file is Safeguard protected) results.
- If a user ID other than the super ID (255, 255) tries to revoke the license for a privileged program (by omitting all *secure-option*s from the command), you receive file-system error 48 (security violation).
- Use the PARTONLY option with REVOKE:
  - To revoke an attribute of a secondary partition named in *fileset-list*. Otherwise, the REVOKE command fails with file-system error 72 (attempt to access an unmounted partition).
  - To revoke an attribute of the primary partition only. Otherwise, FUP revokes the attribute for all partitions of the file.
- REVOKE cannot work with SQL files other than SQL object files.

REVOKE (Super ID) Examples

- To revoke the CLEARONPURGE attribute (for the primary partition only), you must own the partitioned file PARTFILE (in the current default volume and subvolume):

```
-REVOKE PARTFILE, PARTONLY, CLEARONPURGE
```

- For the super ID (255,255) to revoke the license of MYPROG in subvolume $SYSTEM.SYSTEM:

```
-REVOKE $SYSTEM.SYSTEM.MYPROG
```

Commands Related to REVOKE (Super ID)

COMMAND	Function	Page
LICENSE	Sets licensed attribute of a file	2-115
SECURE	Sets file security and other attributes of a file	2-161

SECURE

Sets or changes the standard security attributes of a file. You must secure files protected by the Safeguard product through its command interpreter (SAFECOM). This command applies only to Enscribe files.

To execute the SECURE command, you must be the owner of the file you want to secure, or you must log on as the super ID (255,255).

```
SECURE fileset-list [ , [ security ]
       [ , secure-option  ]...


security is:

   [ " ] { security-string } [ " ]
         { security-num    }


secure-option is:

   CLEARONPURGE | PARTONLY | PROGID
```

fileset-list

is a list of files whose security attributes are to be set or changed. Partial file names are expanded using the current default node, volume, and subvolume (if necessary). You can use wild-card characters and specify *qualified-fileset* for *fileset-list*.

security-string

sets new Guardian file security for the files in *fileset-list*. You can also enclose *security-string* with quotation marks. Specify *security-string* as a literal four-character string. You can include the ? character, but you must always supply all four (rwep) security characters:

rwep

The four characters in *security-string* assign these new values for file security:

rwep

r	Read access allowed
w	Write access allowed
e	Execute access allowed
p	Purge access allowed

You can use these characters in *security-string*:

-	Local super ID (255,255) only (use *security-num* to set in *r* position)
O	Owner only (local only)
G	Group member or owner (local only)
A	Any local user
U	Member of owner's user class (local or remote user with the same user ID as the owner of the file)
C	Member of owner's community (local or remote user with the same group ID as the owner of the file)
N	Any local or remote user
?	No change

Note. You must supply all four (*rwep*) security characters.

security-num

is an integer encoding of the file security. You cannot specify the (?) character attribute in *security-num* form. You can specify *security-num* in octal notation (% indicates octal notation) as in:

%ijkkkk

i	1 if the PROGID option is to be set; 0 if it is not.
j	4 if CLEARONPURGE is to be set; 0 if it is not.

You can also specify the decimal equivalent of the octal number. See the PROGID and CLEARONPURGE options in this syntax description. You cannot specify the ? character attribute in *security-num* form.

The *kkkk* sets the values for read, write, execute, and purge, respectively. *k* can be any of:

0	Any local user
1	Member of owner's group
2	Owner
4	Any local or remote user
5	Member of owner's community
6	Member of owner's user class
7	Local super ID (255,255) only

A standard user can secure a file (by using *security-num*) to allow only super ID (255,255) access (%7777). After doing this, the user does not have access to the file and must ask the super ID to change the security of the file.

Note. The SECURE command does not change the file security if you omit *security-string* and *security-num*.

CLEARONPURGE

physically deletes all data in *fileset-list* from the disk (by overwriting the file space with blank data) when the file is purged (or deallocated) with the DEALLOCATE command. When you purge a file that does not have CLEARONPURGE set, the disk space is logically deallocated, and the data is not physically destroyed.

CLEARONPURGE does not affect the PURGEDATA command, but it does affect the DEALLOCATE command. If the CLEARONPURGE option is specified in a SECURE command for a file, a subsequent DEALLOCATE command physically clears the data from the deallocated extents of the file.

PARTONLY

changes security only for the designated partition (for partitioned files). If you omit PARTONLY, security for every partition of the file is affected if the primary partition of the file is included in *fileset-list*. If the primary partition is not included in *fileset-list*, only the referenced file partitions are affected.

PROGID

is the program ID for program files only. When the program is run, the PROGID option sets the process accessor ID to the ID of the owner of the program file. This option is not valid if you are on a remote system.

SECURE Guidelines

- To secure Safeguard protected files, unless you are a super ID (255,255), you must use the Safeguard command interpreter (SAFECOM). If you try to use FUP to secure a Safeguard protected file, you receive file-system error 199 (disk file is Safeguard protected).
- The owner, group manager, or super ID (255,255) can secure a file. Remote members of the owner’s user class can also secure files if they have read and purge access to them. If you try to use FUP to secure a file without having read and purge security access to it, you receive file-system error 48 (security violation).
- You cannot secure a file that is open with exclusive access.
- If a process has a file open when you secure it, the access rights of the process are not affected until the process closes the file.

- FUP SECURE changes the security of existing files. When you create a new file, its security is defined by your default Guardian file security. To change it, use the DEFAULT command at the TACL prompt.
- FUP SECURE cannot secure SQL files other than SQL object files. To change the security of other SQL files, use the SQLCI SECURE command.
- If you use SECURE against SQL-compiled objects, you receive error 197 (an SQL error has occurred).
- To reset security attributes, use the REVOKE command.
- The *security-string* cannot begin with a hyphen (-). To set the read access to the local super ID (255,255) only, specify a *security-num* (for example, %7*jjj*).

SECURE Examples

- To change the security for MYFILE (a file in the current default subvolume) to let any local user read the file (but only the owner can write, execute, or purge the file):

```
-SECURE MYFILE, "AOOO"
```

- To make the same change, but use the numeric notation to designate the security:

```
-SECURE MYFILE, %0222
```

- To change the security for MYFILE to let any local or remote user read the file, but keep all the remaining security attributes:

```
-SECURE MYFILE, N???
```

- To secure MYPROG, permit only the owner to read, write, and purge it (although any local user can execute it), and set the PROGID bit so the owner ID of MYPROG is used as the process accessor ID when the program is run:

```
-SECURE MYPROG, OOAO, PROGID
```

- To set the network security for the file $OFFICE.BILLS.PAPER so any local or remote user can read the file, only the local group members can execute the file, and only the local owner can write or purge the file:

```
-SECURE $OFFICE.BILLS.PAPER, NOGO
```

- To resecure all the files owned by WRITE.KIRK with NNNN security:

```
-SECURE * WHERE OWNER=WRITE.KIRK, NNNN
```

Commands Related to SECURE

COMMAND	Function	Page
REVOKE	Resets file security and other attributes of a file	2-159

SET

Changes one or more file-creation default attributes before you create files. You can specify parameter values explicitly or set them to match those of an existing file.

To display the current file-creation attributes (the values for *create-param*), use the SHOW command. To restore *create-param* values to the default settings, use the RESET command.

```
SET create-param [ , create-param ] ...


create-param for all file types is:

   [ NO ] AUDIT
   CODE file-code
   EXT { extent-size                              }
       { ( pri-extent-size , sec-extent-size ) }
   FORMAT formatcode
   LIKE filename
   [ NO ] REFRESH
   TYPE file-type


create-param for all structured files is:

   BLOCK data-block-length
   REC record-length


create-param for key-sequenced files is:

   [ NO ] AUDITCOMPRESS
   [ NO ] BUFFERED
   [ NO ] COMPRESS
   [ NO ] DCOMPRESS
   [ NO ] ICOMPRESS
   KEYLEN key-length
   KEYOFF key-offset
   MAXEXTENTS maximum-extents
   QUEUEFILE
   [ NO ] SERIALWRITES
   [ NO ] VERIFIEDWRITES


create-param for partitioned files is:

   PART ( sec-partition-num , [\node.]$volume
      [ , pri-extent-size [ , [ sec-extent-size ]
         [ , partial-key-value ] ] ] )
   [ NO ] PARTONLY
```

create-param for files with alternate-key fields is:

```
[ NO ] ALTCREATE
ALTFILE ( key-file-number , filename )
ALTKEY ( key-specifier { , altkey-param }... )
```

where *altkey-param* is:

```
FILE key-file-number
[ NO ] INSERTIONORDER
KEYLEN key-length
KEYOFF key-offset
NO NULL
NULL null-value
[ NO ] UNIQUE
[ NO ] UPDATE
```

create-param for unstructured files is:

```
BUFFERSIZE unstructured-buffer-size
ODDUNSTR
```

create-param for files on SMF virtual disks is:

```
PHYSVOL [ physvol ]
```

SET Parameters for All File Types

The available value of *create-param* depends on the type of file you want to create. These options are available for all file types:

`[ NO ] AUDIT`

designates the file as audited or nonaudited by TMF. The default is NO AUDIT.

`[ NO ] AUDITCOMPRESS`

sets the mode of producing audit-checkpoint messages (using compressed or entire messages) for audited files. The default is NO AUDITCOMPRESS.

`[ NO ] BUFFERED`

sets the mode of handling write requests to the file using buffered or write-through cache. A buffered cache is specified with BUFFERED. A write-through cache is

specified with NO BUFFERED. The default is BUFFERED for audited files and NO BUFFERED for nonaudited files.

△ **Caution.** If you use the buffered-cache option on a DP2 file that is not audited by TMF, a system failure or disk-process takeover can cause the loss of buffered updates to the file. An application program might not detect this loss (or handle the loss correctly) unless it is modified to do so.

`CODE` *`file-code`*

sets the file code. Specify *`file-code`* as an integer in the range 0 through 65,535. The default is zero. File codes 100 through 999 are reserved for use by HP.

Note. For a list of the reserved file codes, see Table 2-2 on page 2-87.

```
EXT { extent-size                          }
    { ( pri-extent-size , sec-extent-size ) }
```

sets the extent size—including primary and secondary extent sizes (if applicable). Extents should be at least as large as the block size (for structured files). If they are not, multiple extents are allocated every time a block is required. The extent size must be an integral multiple of the unstructured buffer size (for unstructured DP2 files).

The default value for *`extent-size`* and *`pri-extent-size`* is one page (2,048 bytes). If you specify a value of zero or do not specify a value, *`sec-extent-size`* defaults to the *`pri-extent-size`* value. When creating files, you cannot define primary or secondary extents as zero pages.

For unstructured files on a disk drive in a disk drive enclosure, the *`pri-extent-size`* and the *`sec-extent-size`* must both be divisible by 14. DP2 automatically rounds the size up if the file is non-partitioned.

You can specify these values for *`extent-size`*, *`pri-extent-size,`* and *`sec-extent-size`*:

`0:`*`maximum`* `[ PAGE[S] ]`

specifies the extent size in pages (2,048-byte units). Possible values of *`maximum`* are:

Format 1	Format 2
`0:65,535 [ PAGE[S] ]`	`0:512,000,000 [ PAGE[S] ]`

Because the minimum extent size is one page (2,048 bytes), one page is also allocated if you specify zero extents.

Note. If you specify an extent size over 65,535 pages, you must assign Format 2 to your files.

`0:`*`maximum`* `BYTE[S]`

specifies the extent size in bytes. Possible values of *`maximum`* are:

Format 1	Format 2
`0:134,215,680 BYTE[S]`	`0:2,147,483,647 BYTE[S]`

The FUP process rounds the extent size up to the next full page. If you specify 2,047 bytes, FUP allocates one page. If you specify 2,049 bytes, FUP allocates two pages.

`0:`*`maximum`* `REC[S]`

specifies the extent size based on the current settings for *`record-length`* (REC), *`data-block-length`* (BLOCK), *`index-block-length`* (IBLOCK), key-field lengths, and compression settings. Possible values are:

Format 1	Format 2
`0:134,215,680 REC[S]`	`0:2,147,483,647 REC[S]`

The FUP process rounds the extent size up to the next full page.

`0:`*`maximum`* `MEGABYTE[S]`

specifies extent sizes in million-byte units. Possible values are:

Format 1	Format 2
`0:134 MEGABYTE[S]`	`0:2,147 MEGABYTE[S]`

The FUP process rounds the extent size up to the next full page.

`FORMAT` *`format-code`*

is the format designator for the new file. The designator can have these values:

0 The system decides the format of the file based on the values of other attributes, such as block size. The system chooses Format 2 when the partition size is over 2 GB minus 1 MB. For files that are not key-sequenced, the total size is 4 GB or more.

1 The file should be Format 1. For more information, see Handling File Formats on page 1-22.

2 The file should be Format 2. For more information, see Handling File Formats on page 1-22.

If the FORMAT option is omitted, it defaults to 0.

`LIKE` *`filename`*

sets file-creation attributes to match those of an existing file. FUP expands a partial file name using the current default values for node, volume, and subvolume (if necessary). If you specify a secondary partition of a partitioned file in *`filename`*, FUP automatically sets the PARTONLY *`create-param`*.

MAXEXTENTS *maximum-extents*

sets the maximum number of extents to be allocated (for nonpartitioned files or key-sequenced partitioned files). Specify *maximum-extents* as an integer from 16 through *n*, where *n* is a maximum value determined by the amount of free space in the file label. The FUP process rounds any value you set (from 1 through 15) up to 16. The absolute maximum is 978 extents, and the default is 16.

It is not always possible to allocate all the extents specified by *maximum-extents*. The actual number of extents that can be allocated depends on the amount of space in the file label. If there are alternate keys or partitions, the maximum number of extents allowed is less than 978. If you specify MAXEXTENTS, you must also consider the primary and secondary extent sizes to avoid exceeding the maximum file size.

When the primary and secondary extent size (plus the specified MAXEXTENT size) is larger than two gigabytes, a CREATE operation rejects the request with file-system error 21 (illegal count specified).

[NO] REFRESH

causes the file label to be copied to disk whenever the file control block is marked as dirty. This situation occurs if the end of file or a free-space block changes. If the only change is updating the LAST MODIFIED field, the file label is not written to disk. The default is NO REFRESH (the label is not copied to disk).

[NO] SERIALWRITES

sets the mode of writes to the mirror: serial or parallel. The default is NO SERIALWRITES.

TYPE *file-type*

sets the file type. The default is U. Values for *file-type* are:

U or 0	Unstructured file
R or 1	Relative file
E or 2	Entry-sequenced file
K or 3	Key-sequenced file

[NO] VERIFIEDWRITES

sets the mode of file writes: verified or unverified. The default is NO VERIFIEDWRITES.

SET Parameters for All Structured Files

The *create-param* available depends on the type of file you want to create. These options are available for all structured files:

BLOCK *data-block-length*

sets the data-block length. Specify *data-block-length* as an integer that is a power of 2 from 512 bytes through 4096 bytes (512, 1024, 2048, 4096).

The default *data-block-length* is 4096 bytes. Blocks should not be larger than extents. If this situation occurs, multiple extents are allocated every time a block is required.

When you are setting file-creation attributes, you can SET BLOCK *data-block-length* to any value within the valid range for this parameter, but FUP rounds it up to a DP2 block size.

You might need to decrease the number of 1 KB cache blocks and increase the number of 4 KB cache blocks (in the DP2 configuration) if there is significant use of the default block size (4096 bytes).

During the file-creation process, DP2 rounds up the extent size (to 2 pages or 4,096 bytes) because the extent size of DP2 files must always be an integral multiple of the BUFFERSIZE (for unstructured files) or of the BLOCK size (for structured files).

REC *record-length*

sets the record length. Specify *record-length* as an integer in the range 1 through:

File Format	Maximum Value for Relative and Entry-Sequenced Files	Maximum Value for Key-Sequenced Files
1	Blocksize - 24	Blocksize - 34
2	Blocksize - 48	Blocksize - 56

The default setting for *record-length* is 80 bytes, and the maximum record length is reduced by one byte when data compression is used.

SET Parameters for Key-Sequenced Files

The *create-param* available depends on the type of file you want to create. These options are available for key-sequenced files:

[NO] COMPRESS

sets or clears the states of key compression in both index and data blocks. The default setting is NO COMPRESS. The key offset must be 0 for key compression in data blocks, and the maximum record size is reduced by one byte.

`[ NO ] DCOMPRESS`

sets or clears key compression for data blocks. The default setting is NO DCOMPRESS. The key offset must be 0 for key compression, and the maximum record size is reduced by one byte.

`[ NO ] ICOMPRESS`

sets or clears key compression in index blocks. The default setting is NO ICOMPRESS.

`KEYLEN` *`key-length`*

sets the primary-key length. Specify *`key-length`* as an integer in the range 1 through 255. To create key-sequenced file structures, you must specify KEYLEN, or the creation attempt fails.

`KEYOFF` *`key-offset`*

sets the primary-key offset. Specify *`key-offset`* as an integer in the range 0 through 2034. The default setting for *`key-offset`* is 0.

`QUEUEFILE`

sets the queue file attributes. Queue files are a special type of key-sequenced file with timestamps automatically added by DP2. Processes can queue and dequeue records in a queue file. Queue files contain variable-length records that are accessed by values in designated key fields. Unlike other key-sequenced files, queue files have primary keys but cannot have alternate keys. The primary key for a queue file includes an 8-byte timestamp. You can optionally add a user key. The disk process inserts the timestamp when each record is inserted into the file and maintains the timestamp during subsequent file operations.

SET Parameters for Partitioned Files

The available value of *`create-param`* depends on the type of file you want to create. These parameter values are described in section SET Parameters for All File Types on page 2-166. These options are available for partitioned files:

```
PART ( sec-partition-num , [\node.]$volume
      [ , pri-extent-size [ , [ sec-extent-size ]

      [ , partial-key-value  ] ] ] )
```

`PART`

sets secondary partition specifications for partitioned files. Specify each secondary partition separately.

sec-partition-num , *node*.$*volume*

names the volume where this secondary partition is to reside. Specify *sec-partition-num* as an integer from 1 through 15 to designate the secondary partition. Specify *node* and *volume* as the names of the node and volume to contain the partition. The file name and the subvolume of the primary partition are specified when the file is created.

Although FUP lets you specify any number in the range of 1 through 15 for *sec-partition-num*, FUP changes it to a standard DP2 number that starts at zero when the file is created.

Note. Example 2-6, DETAIL Format for SQL Tables and Indexes and for Enscribe and OSS Files, on page 2-98 shows the DP2 number as listed in the INFO DETAIL command.

---

pri-extent-size , *sec-extent-size*

sets the primary and secondary extent sizes. The default value for *pri-extent-size* is one page (2,048 bytes). If you specify a value of zero or do not specify a value, *sec-extent-size* defaults to the *pri-extent-size* value. When altering files, you cannot define primary or secondary extents as zero pages.

For partitioned unstructured files where one or more partitions reside on a disk drive in a disk drive enclosure, DP2 has additional restrictions. You must specify both the *pri-extent-size* and the *sec-extent-size* so that they can be explicitly divisible by 14. DP2 does not automatically round the size up.

You can specify these values for *pri-extent-size* and *sec-extent-size*:

0:*maximum* [PAGE[S]]

specifies the extent size in pages (2,048-byte units). Possible values are:

Format 1	Format 2
0:65,535 [PAGE[S]]	0:512,000,000 [PAGE[S]]

Because the minimum extent size is one page (2,048 bytes), one page is also allocated if you specify zero extents.

0:*maximum* BYTE[S]

specifies the extent size in bytes. Possible values are:

Format 1	Format 2
0:134,215,680 BYTE[S]	0:2,147,483,647 BYTE[S]

The FUP process rounds the extent size up to the next full page. If you specify 2,047 bytes, FUP allocates one page. If you specify 2,049 bytes, FUP allocates two pages.

`0:`*`maximum`* `REC[S]`

specifies the extent size based on the current settings for *`record-length`* (REC), *`data-block-length`* (BLOCK), *`index-block-length`* (IBLOCK), key-field lengths, and compression settings. Possible values are:

Format 1	Format 2
`0:134,215,680 REC[S]`	`0:2,147,483,647 REC[S]`

The FUP process rounds the extent size up to the next full page.

`0:`*`maximum`* `MEGABYTE[S]`

specifies extent sizes in million-byte units. Possible values are:

Format 1	Format 2
`0:134 MEGABYTE[S]`	`0:2,147 MEGABYTE[S]`

The FUP process rounds the extent size up to the next full page.

`partial-key-value`

specifies the lowest key value that can reside in this partition (for key-sequenced files only). You must include *`partial-key-value`* for each partition of a key-sequenced file. The partial-key value is 0 for the primary partition.

Specify *`partial-key-value`* as any one (or a combination) of:

- A string of characters enclosed in quotation marks:

  `"`*`c1c2...cn`*`"`

- A list of single characters (with quotation marks around each character) separated by commas:

  `"`*`c`*`" , "`*`c`*`" , "`*`c`*`" ...`

- Integers representing byte values from 0 through 255 enclosed in brackets:

  `[ 5, 8, 220 ]`

To specify *`partial-key-value`* correctly:

- If the partial-key values are a string of alphabetic characters, enclose them in quotation marks. For example, enter A, B, and C as a string of characters:

  `"ABC"`

- If the partial-key values include a string of integers only, each item is separated by a comma, and the entire string must be enclosed in brackets:

  `[ 1,2,3 ]`

- If the partial-key values include both single characters and integers:
  - The entire string must be surrounded by brackets.
  - Each single alphabetic character in the string must be enclosed in quotation marks.
  - Quotation marks are not used for integers.
  - Each item in the string must be separated by a comma.

  For example, enter a string consisting of A, 22, and C as:

  ```
  [ "A", 22, "C" ]
  ```

`[ NO ] PARTONLY`

specifies whether subsequent file creations create all partitions of a partitioned file (NO PARTONLY) or a single partition (PARTONLY). The default setting is NO PARTONLY.

If you specify PARTONLY while a PART specification is enabled, any file you create is designated as a primary partition. If a PART specification is not enabled, any file you create is designated as a secondary partition. When you create the file, you must specify the file name of the (primary or secondary) partition.

SET Parameters for Files With Alternate-Key Fields

The *create-param* available depends on the type of file you want to create. These options are available for files with alternate-key fields:

`[ NO ] ALTCREATE`

sets or clears the automatic alternate-key file creation. The alternate-key files are created (if you specify ALTCREATE) when you create the primary file. The default setting is ALTCREATE.

`ALTFILE ( key-file-number, filename )`

adds or replaces the file name of an alternate-key file. You must include this parameter for any undefined key file number that is referenced by an ALTKEY specification. Specify *key-file-number* as an integer in the range 0 through 255 to designate the alternate-key file being named.

The FUP process expands a partial file name by adding the current default names for node, volume, and subvolume.

If you add a new ALTFILE number, your SET command must also include an ALTKEY option that specifies the alternate-key file.

If you are defining a new alternate-key file, you must create the file in a separate operation either before or after using SET ALTFILE.

If you are deleting the last alternate-key specification, you must also delete the corresponding alternate-file specification in the same SET command.

If you are deleting the last alternate-file specification, you must also delete the corresponding alternate-key specification in the same SET command.

ALTKEY (*key-specifier* { , *altkey-param* }...)

sets an alternate-key specification. You must specify each alternate key separately.

key-specifier

is a 2-byte value that identifies this alternate-key field. Specify *key-specifier* as either a one-character or two-character string in quotation marks:

"[*c1*]*c2*"

Or specify it as an integer from -32,768 through 32,767:

{ -32,768 : 32,767 }

You can use any characters for *key-specifier* except zero. If you omit *c1*, then *c1* is treated as a zero.

Note. If you add a new key specifier that references an undefined key-file number, you must include the ALTFILE option to define the alternate-key file.

---

altkey-param

specifies attributes of the alternate-key file.

FILE *key-file-number*

sets the key-file number for *key-specifier*. Specify *key-file-number* as an integer from 0 through 255. ALTFILE *create-param* relates this number to an actual file. The default is zero.

[NO] INSERTIONORDER

specifies whether or not insertion-ordered alternate-key sequencing is to be used. The default is NO INSERTIONORDER. The default specifies alternate key records of files with duplicate key values ordered by their primary key sequence and not their order of insertion.

An insertion-ordered alternate key cannot share an alternate-key file with other keys of different lengths or with other keys that are not insertion-ordered.

All nonunique alternate keys of a file must have the same duplicate-key ordering attribute. A file with this specification must not have both insertion-ordered alternate keys and standard (duplicate ordering by primary key) nonunique alternate keys.

KEYLEN *key-length*

sets the key length for *key-specifier*. You must specify a KEYLEN to create a key-sequenced file, or the creation attempt fails.

KEYOFF *key-offset*

sets the key offset for *key-specifier*. The default setting for *key-offset* is zero.

NO NULL | NULL *null-value*

specifies whether or not a null value is set for *key-specifier*. If a value is specified, *null-value* must be an ASCII character in quotation marks or an integer in the range 0 through 255. The default is NO NULL.

Note. For information about null values, see the *Enscribe Programmer's Guide.*

[NO] UNIQUE

specifies whether or not *key-specifier* is set as a unique key. The default is NO UNIQUE.

[NO] UPDATE

specifies whether or not automatic updating is set for the alternate-key file represented by *key-specifier*.

The NO UPDATE option prevents the file system from automatically updating the specified alternate-key file when you write to the main file. Although you usually keep alternate-key files synchronized with their main files, you might leave files unsynchronized in rare circumstances. For example, you could have two files pointing to the same alternate-key file but only want updates from one of the two written to it.

The default is UPDATE.

SET Parameters for Unstructured Files

The *create-param* set available depends on the type of file you want to create. These options are available for unstructured files:

BUFFERSIZE *unstructured-buffer-size*

is the internal buffer size to use when accessing the specified file (for unstructured files only). To set the BUFFERSIZE file attribute, use this command (FUP SET) or the FUP ALTER and FUP CREATE commands. Possible values for *unstructured-buffer-size* are (in bytes):

512, 1024, 2048, 4096

FUP rounds the actual buffer size up to the nearest valid DP2 block size.

During the file-creation process, DP2 rounds up the extent size (to 2 pages or 4,096 bytes) because the extent size of DP2 files must always be an integral multiple of the BUFFERSIZE (for unstructured files) or of the BLOCK size (for structured files).

To create an unstructured DP2 file with one-page extents, you must specify a BUFFERSIZE of 2048 bytes with either the FUP SET or FUP CREATE command.

`ODDUNSTR`

changes an even unstructured file to an odd unstructured file.

Unstructured Enscribe files can be even or odd. The FUP process rounds up any odd byte count that you give to an even unstructured file (for reading, writing, or positioning). This is the default for unstructured files.

FUP does not round up odd unstructured files. You always read, write, or position at the byte count you give. To change an odd unstructured file to an even unstructured file, copy the odd file into a new file that was created as even unstructured.

SET Parameter for Files on SMF Virtual Disks

This option is available only for SMF files (all other parameters are valid for SMF files):

`PHYSVOL [ physvol ]`

specifies the physical volume where a file should be created. This option overrides any SMF parameters after the CREATE command creates a file on the virtual disk. The value of *physvol* specified must be included in the storage pool associated with the SMF virtual disk process.

SET Guidelines

- The SET command sets the file-creation attributes only for the current session of FUP. Each time you start a new FUP session (when you type FUP at the command interpreter prompt), the attributes are reset to their default values. You must use FUP in the interactive mode to use the SET and SHOW features.
- The REFRESH, DCOMPRESS, ICOMPRESS, and COMPRESS attributes are not passed to alternate-key files when you create the primary-key file. To set these attributes, include the NO ALTCREATE option and create the alternate-key files in a separate operation.
- If you use the PHYSVOL option with the SET command, it applies to each subsequent CREATE command until it is reset. An error occurs during a CREATE command if the PHYSVOL specified does not belong to the volume specified in the CREATE command.
- The extent size is rounded up when you create a file according to these rules:
  - The extent size must be a multiple of the block size (for DP2 structured files).

- The extent size must be a multiple of the unstructured buffer size (for DP2 unstructured files).

Note. For information on extent-size rounding, see the *Enscribe Programmer's Guide.*

Table 2-3 summarizes the upward rounding that can occur at file creation. Block and buffer sizes are in bytes, and extent sizes are in pages. (The default extent size is one page.) Although you can specify different sizes for primary and secondary extents, equal sizes are displayed to simplify the table.

Table 2-3. Extent-Size Rounding

Type of File	EXT Parameter Value	Extent Size Created	
DP2 Files	Unstructured File	Any N	(N, N)
	With BUFFERSIZE <= 4096	Even N	(N, N)
	With BUFFERSIZE > 4096	Odd N	(N+1, N+1)
	With BUFFERSIZE > 4096	Default	(2, 2)
	With BUFFERSIZE > 4096		
	Structured File	Default	(1, 1)
	With BLOCK <= 4096	Any N	N, N)
	With BLOCK <= 4096	Even N	(N, N)
	With BLOCK > 4096	Odd N	(N+1, N+1)
	With BLOCK > 4096		

SET Examples

- To create a non-partitioned unstructured file on a disk drive in a disk drive enclosure:

```
-SET TYPE U
-SET EXT (2,20)
-SET REC 80
-CREATE TEST
```

Note. DP rounds the extent sizes up to multiples of 14 so the actual extent information will be ext (14,28), not ext (2,20) as specified.

- To create a partitioned unstructured file where one or more partitions reside on a disk drive in a disk drive enclosure.

```
-SET TYPE U
-SET EXT (14,42)
-SET REC 80
-SET PART (1,$FIBRE,14,42)
-CREATE TESTPART
```

Note. DP2 requires that the *pri-extent-size* and *sec-extent-size* of partitioned unstructured files explicitly be multiples of 14. It further requires that the pri-extent-size for every partition be the same; the *sec-extent-size* for every partition must also be the same for every partition. If the extent size is not the same for every partition, error 21 is returned.

- If you set the BUFFERSIZE for an unstructured DP2 file, FUP rounds the buffer size up to the next DP2 block size. Suppose you set the buffer at 30 bytes:

  ```
  -SET BUFFERSIZE 30
  ```

  The SHOW command indicates the buffer size is 512 bytes:

  ```
  -SHOW
  TYPE U
  FORMAT 1
  EXT ( 1 PAGES, 1 PAGES )
  MAXEXTENTS 16
  BUFFERSIZE 512
  ```

- To set file-creation attributes for a key-sequenced file with 50-byte records and a primary-key length of 36 bytes:

  ```
  -SET TYPE K, REC 50, KEYLEN 36
  ```

- To create the file $L.SMS.BLUE on the physical volume $ABC, use the PHYSVOL option of the SET command (which subsequently resets the PHYSVOL option):

  ```
  -SET PHYSVOL $ABC
  CREATE $L.SMS.BLUE
  RESET PHYSVOL
  ```

Commands Related to SET

COMMAND	Function	Page
CREATE	Creates a file using default file attributes	2-57
RESET	Resets default file attributes or CONFIG[URE] options	2-155
SHOW	Displays default file attributes or CONFIG[URE] options	2-180

SHOW

Displays the current settings of the file-creation attributes. Use the FUP SET and FUP RESET commands to set and reset these attributes.

```
SHOW [ / OUT listfile / ]  [ show-opts ]  [ configure-opts ]


show-opts is:

create-spec [ , create-spec ] ...

create-spec is:

TYPE
CODE
FORMAT
EXT
REC
BLOCK
COMPRESS
DCOMPRESS
ICOMPRESS
KEYLEN
KEYOFF
ALTKEY [ key-specifier ]
ALTKEYS
ALTFILE [ key-file-number ]
ALTFILES
ALTCREATE
ODDUNSTR
REFRESH
AUDIT
PART [ partition-num ]
PARTS
PARTONLY
PHYSVOL
MAXEXTENTS
BUFFERSIZE
BUFFERED
AUDITCOMPRESS
VERIFIEDWRITES
SERIALWRITES
QUEUEFILE


configure-opts is:

[ CONFIG[URE] [ AS COMMANDS ] [ config-simple-opts ] ]
```

```
config-simple-opts
  config-simple-option [ , config-simple-option ]

config-simple-option is:

ALLOW
DISPLAYBITS
ECHO [CONFIG[URE] | OBEY
IOTIMEOUT
NETBLOCKSIZE
PROMPT [ PURGE ]
REPORTWIDTH
RESTARTUPDATE
STATONLY
XLATE [ xlate-table-name ]
XLATEIN [ xlate-table-name ]
XLATEOUT [ xlate-table-name ]
COPY [ copy-option ]
DUP[LICATE] [ dup-option ]
LOAD [ load-option ]
LOADALTFILE [ loadaltfile-option ]
RELOAD [ reload-option ]
```

OUT *listfile*

names an existing disk file or a device to receive the listing output from the SHOW command. You can use either a standard file name or a spool DEFINE name as the OUT *listfile* for a SHOW command. If *listfile* is an existing file, FUP appends the output to the file.

Note. For more information about *listfile*, see Specifying Files on page 1-8.

create-spec

specifies which file attributes you want to show. If you do not specify any attributes, FUP shows all the current settings applicable to the current file type.

Note. For *create-spec* parameter default values, see SET on page 2-165. For *config-simple-opts*, see CONFIG[URE] on page 2-26.

AS COMMANDS

causes FUP to insert the word *CONFIGURE* in front of the CONFIG[URE] options. If this option is present, it can only occur immediately after the CONFIG[URE] keyword, and you must specify both words. If this display is written to a file, you can use it to reestablish CONFIG[URE] options from a command file or FUPCSTM file.

SHOW Guidelines

- You cannot use *show-opts* and *configure-opts* in the same command. You must use separate SHOW commands.

- If you issue the SHOW command with a *create-spec* that is not currently set (or is not applicable to the current value of TYPE), FUP returns only a prompt.
- The SHOW display might list an EXT size that differs by one page from what might actually be created. This situation occurs because FUP rounds extent sizes up (if necessary).

Note. For more information about the EXT parameter, see SET on page 2-165.

- Use the DISPLAYBITS option (from the CONFIGURE command) with SHOW for a file that has alternate keys containing 8-bit characters in the partial key-value field. For more information about the DISPLAYBITS option, see CONFIG[URE] on page 2-26.
- You can set the extent size as the number of records in each extent. SHOW displays the extent size in that form. For example, suppose you enter:

  ```
  -SET EXT (100 RECS, 10)
  ```

  The SHOW display then includes:

  ```
  EXT ( 100 RECS, 10 PAGES )
  ```

- Use the SHOW command with the PHYSVOL option to display any SMF settings for the file-creation attributes.

SHOW Examples

- This example assumes you enter FUP SHOW when the defaults are enabled for all file-creation attributes, causing this information to be displayed:

  ```
  TYPE U
  FORMAT 1
  EXT ( 1 PAGES, 1 PAGES )
  MAXEXTENTS 16
  BUFFERSIZE 4096
  ```

- To use the SHOW command to display the result of the SET commands used to assign specific file-creation attributes:

  ```
  > FUP
  -SET TYPE K
  -SET KEYLEN 2
  -SET ALTKEY ("aa",FILE 0,KEYLEN 2,KEYOFF 0,INSERTIONORDER)
  -SET ALTFILE (0, $DATAA.DCDTEST.ALT0)
  -SHOW
  TYPE K
  EXT ( 1 PAGES, 1 PAGES )
  FORMAT 1
  REC 80
  BLOCK 4096
  KEYLEN 2
  KEYOFF 0
  ALTKEY ( "aa", FILE 0, KEYOFF 0, KEYLEN 2, INSERTIONORDER)
  ALTFILE ( 0, $DATAA.DCDTEST.ALT0)
  ```

```
ALTCREATE
MAXEXTENTS 16
-EXIT
```

- Suppose these CONFIG[URE] commands are enabled:

  ```
  CONFIGURE NETBLOCKSIZE 28
  CONFIGURE DUP SOURCEDATE
  ```

  A SHOW CONFIG[URE] command provides this result:

  ```
  ALLOW ABENDS OFF
  NO ECHO CONFIGURE
  ECHO OBEY
  NO PROMPT PURGE
  NETBLOCKSIZE 28
  DUP SOURCEDATE
  ```

Commands Related to SHOW

COMMAND	Function	Page
CONFIG[URE]	Sets some of the attributes displayed by SHOW	2-26
CREATE	Creates a file using default file attributes	2-57
RESET	Resets default file attributes or CONFIG[URE] options	2-155
SET	Sets default file attributes	2-165

STATUS

Reports the status of a reload operation.

```
STATUS [ / OUT listfile / ] filename [ , DETAIL ]
```

`OUT` *`listfile`*

names an existing disk file or a device to receive the listing output from the STATUS command. You can use either a standard file name or a spool DEFINE name as the OUT *`listfile`* for a STATUS command. If *`listfile`* is an existing file, FUP appends the output to the file.

Note. For more information about *`listfile`*, see Specifying Files on page 1-8.

`filename`

is the name of a key-sequenced file or SQL object reorganized by a reload operation. You cannot use wild-card characters in *`filename`* or specify *`qualified-fileset`*.

Note. When using SPI interface, pass the filename in uppercase.

DETAIL

gives complete information on the status of the file you want listed. By default, STATUS displays information only for fields it determines are relevant to the current situation. For example, if a RELOAD process finishes, the STATUS command displays only the completion time unless the DETAIL option is specified.

STATUS Guidelines

- If the reload operation is complete, FUP displays:

```
$vol-name.subvol-name.filename
RELOAD COMPLETED: date-time of completion
```

- If the reload operation is suspended (by the SUSPEND command), FUP displays:

```
$vol-name.subvol-name.filename
  RELOAD INITIATED: date-time of initiation
         SUSPENDED: date-time of suspension
            DSLACK: percentage%
            ISLACK: percentage%
              RATE: percentage%
     PCT COMPLETED: percentage%
        DEALLOCATE: [YES|NO]
             SHARE; [YES|NO]
           RECLAIM: [YES|NO]
```

- If the reload operation is in progress, FUP displays:

```
$vol-name.subvol-name.filename
RELOAD IN PROGRESS
  RELOAD INITIATED: date-time of initiation
           RESUMED: date-time resumed
            DSLACK: percentage%
            ISLACK: percentage%
              RATE: percentage%
     PCT COMPLETED: percentage%
        DEALLOCATE: [YES|NO]
```

The FUP process displays the RESUMED date and time only if the reload operation was stopped and then restarted.

STATUS Examples

- To display the status of a RELOAD process while it is in progress:

```
-STATUS PAYFILE
```

```
$BASE.FUPTESTL.PAYFILE
RELOAD IN PROGRESS
  RELOAD INITIATED: 10 Aug 2000, 15:35
            DSLACK: 20%
            ISLACK: 20%
              RATE: 10%
     PCT COMPLETED: 45%
        DEALLOCATE: YES
```

- To display the status of a RELOAD process after its completion:

```
-STATUS PAYFILE, DETAIL

$BASE.FUPTESTL.PAYFILE
  RELOAD INITIATED: 10 Aug 2000, 15:37
  RELOAD COMPLETED: 10 Aug 2000, 15:39
            DSLACK: 0%
            ISLACK: 0%
              RATE: 100%
     PCT COMPLETED: 100%
        DEALLOCATE: YES
```

Commands Related to STATUS

COMMAND	Function	Page
RELOAD	Initiates a RELOAD operation	2-146
SUSPEND	Suspends a RELOAD operation	2-186

SUBVOLS

Displays the names of all the subvolumes on a particular volume.

```
SUBVOLS [ / OUT listfile / ] [ subvolset ]

subvolset is:

   [ \node.] $volume [.subvol ] | subvol
```

`OUT` *`listfile`*

names an existing file or device to receive the output of the SUBVOLS command. You can use either a standard file name or a spool DEFINE name as the OUT *`listfile`* for the SUBVOLS command. If *`listfile`* is an existing file, FUP appends the output to that file.

Note. For more information about *`listfile`*, see Specifying Files on page 1-8.

`subvolset`

is a pattern describing the subvolume names you want to see. The *`subvolset`* parameter defaults to the current volume. You can use wild-card characters in *`subvolset`* (including the volume name), but you cannot specify *`qualified-fileset`*.

SUBVOLS Examples

- To display the names of all the subvolumes on the $HANSEL volume:

```
-SUBVOLS $HANSEL
$HANSEL
    BKUPLIST  CARSELLA  CBLIBERT  D04RGEN   GSPURGE
    JOAND20   LATBOOT   LATCA     LATT001   LATT002
    LATT003   LATT004   LATT005   LATT006   LATT007
    LATT0     LATT013   LATT014   LATT015   LATT016
    LATT017   LATT018   LATT019   LATT0     LUUTALQA
    OSMAC     OSMAC08   PETER     RAUBATS   RAUD00
```

- To use the wild-card option to display the names of a specific set of subvolumes:

```
-SUBVOLS $LARS*.GARY*
$LARS
    GARYOLD   GARYTACL  GARYWORK  GARY1
$LARS2
    GARYNEW   GARYZ
```

SUSPEND

Stops a reload operation. A subsequent RELOAD command can resume the reload operation from the point where it was suspended. For more information, see RELOAD on page 2-146.

```
SUSPEND [ / OUT listfile / ] filename
```

`OUT` *`listfile`*

names an existing disk file or a device to receive the listing output from the SUSPEND command. You can use either a standard file name or a spool DEFINE name as the OUT *`listfile`* for a SUSPEND command. If *`listfile`* is an existing file, FUP appends the output to that file.

Note. For more information about *`listfile`*, see Specifying Files on page 1-8.

`filename`

is the name of a key-sequenced file or SQL object (table or index) being reorganized. You cannot use wild-card characters in *`filename`* or specify *`qualified-fileset`*

Note. When using SPI interface, pass the filename in uppercase.

SUSPEND Guidelines

- When you issue a SUSPEND command, FUP creates an online reload server (ORSERV) process to stop the reload operation (which is controlled by another ORSERV process). If FUP cannot create the second ORSERV process, an error message appears.
- If a reload operation is not in progress for *filename* (or if the ORSERV process cannot be stopped), FUP displays an error message.
- If the reload operation has already been suspended, FUP ignores the SUSPEND command and does not display a message.

SUSPEND Example

```
-SUSPEND MYFILE
```

Commands Related to SUSPEND

COMMAND	Function	Page
RELOAD	Initiates a RELOAD operation	2-146
STATUS	Displays the status of a RELOAD operation	2-183

SYSTEM

Sets the default node. You can also set the default volume and subvolume names used by FUP in expanding file names.

```
SYSTEM [ \node [.$volume ] [.subvolume ] ]
```

node

is a node name.

volume

names the new current default volume.

subvolume

names the new current default subvolume.

SYSTEM Guidelines

- The initial SYSTEM setting is the node enabled when you start FUP.
- You can use the SYSTEM command only to reference nodes that have a name; for example, a node that is part of a network.

- Entering the SYSTEM command without *node* restores the default node name to the local node.
- If you omit *node* parameter, FUP uses the local form for file-name expansion. Using the local form allows access to all volume and device names on the node.
- When you supply *node* (even if the name of the local node is specified), FUP uses the network form for file-name expansion. Using the network form causes only volume or device names with six characters or fewer to be accessible.
- The *volume* and *subvolume* parameters are optional. If you omit these parameters, FUP uses their current default values.
- If *node* is an invalid node name, FUP displays an error message and does not change the node. However, FUP does not issue an error if *volume* (or its default value) does not exist on the new system.
- FUP and the command interpreter keep separate defaults for node, volume, and subvolume. When you exit FUP, the command interpreter defaults are still the values that were enabled when you started FUP.

SYSTEM Examples

- To set the \LONDON node as the current default for FUP and keep your current default volume and subvolume:

  ```
  -SYSTEM \LONDON
  ```

- To set the current FUP defaults to the PAID subvolume on the $BILLS volume of the NEWYORK node:

  ```
  -SYSTEM \NEWYORK.$BILLS.PAID
  ```

Commands Related to SYSTEM

COMMAND	Function	Page
VOLUME	Changes the FUP default volume and subvolume	2-190

TRUST

Changes the state of Trust flag to TRUST ME, TRUST SHARED or TRUST OFF. Only a super ID (255, 255) user can set the FLTTrustFlags flag to TRUST ME or TRUST SHARED. This flag controls whether direct I/O access to user buffers is permitted when this process is running.

```
TRUST fileset-list, trust-flag
```

fileset-list

is a list of disk files whose state of Trust flag is to be changed.

trust-flag

is one of:

```
OFF
ME
SHARED
```

OFF

resets Trust flag.

ME

sets Trust flag to TRUST ME state.

SHARED

sets Trust flags to TRUST SHARED state.

TRUST Guidelines

- The code of the files must be 100 or 800. If a files has another file code, error 2 occurs.
- If a user who is not the super ID (255,255) attempts to change the Trust flags to TRUST ME or TRUST SHARED, FUP gives a warning of INSUFFICIENT CAPABILITY.

TRUST Examples

- To set TRUST flag of file MYPROG to TRUST ME:

  ```
  -TRUST MYPROG, ME
  ```

- To set TRUST flag of file MYPROG to TRUSTSHARED:

  ```
  -TRUST MYPROG, SHARED
  ```

- To rest TRUST flag of file MYPROG:

  ```
  -TRUST MYPROG, OFF
  ```

VOLS

Displays information about volumes on a system. The VOLS command lists the names, capacities, and current usage for all specified volumes in a format that is similar to the Disk Space Analysis Program (DSAP) utility.

```
VOLS [ / OUT listfile / ] [ volset ]

volset is:

   [ \node.] $volume
```

`OUT listfile`

names an existing disk file or a device to receive the listing output from the VOLS command. You can use either a standard file name or a spool DEFINE name as the OUT *listfile* for a VOLS command. If *listfile* is an existing file, FUP appends the output to that file.

Note. For more information about *listfile*, see Specifying Files on page 1-8.

volset

specifies volumes to display information about. The *volset* parameter can be the name of a volume or a wild card, and it can include a system name. If you specify a volume name, it must begin with a dollar sign ($). The default is all volumes on the current system ($*).

VOLS Example

To show information on all volumes of the current system that begin with the letter *M*:

```
-VOLS $M*
                -- Capacity (Mb) --   %     -- Free Extents --
Volume    (M)    Total      Free     Free    Count    Biggest
$MCAT      Y       895    346.62       38       12     332.24
$MOLD      Y       415     50.04       12       25      26.67
$MONO              895    289.97       32      125     287.74
$MONO1             895    314.41       35        1     314.41
$MOSS      Y       415    150.09       36      491      66.67
```

VOLUME

Changes the current default volume or subvolume names used during the execution of FUP. The initial defaults are the subvolume and volume that were the current defaults when you started FUP.

```
VOLUME [ [\node.] $volume [.subvol ] ]
       [ [\node.] subvol              ]
```

node

sets the current default node for FUP. You can set the default node only in a named node; for example, a node that is part of a network.

volume

sets the current default volume name for FUP operations.

subvol

sets the current default subvolume name for FUP operations.

VOLUME Guidelines

- Typing the VOLUME command without a parameter restores the defaults that were enabled when you started FUP, including the original node.
- When you supply *node* (even if you specify the name of the local node), FUP uses the network form for file-name expansion. Using the network form causes only volume or device names with six characters or fewer to be accessible.
- You cannot specify a new current default volume name that has more than six characters after the dollar sign ($) on a remote node.
- *node* and *subvolume* are optional. If you omit these parameters, FUP uses their current default values.
- If *node* is an invalid node name, FUP displays an error message and does not change the node. However, FUP does not issue an error if *volume* (or its default value) does not exist on the new system.
- FUP and the command interpreter keep separate defaults for node, volume, and subvolume. When you exit FUP, the command interpreter defaults are still the values that were enabled when you started FUP.

VOLUME Examples

- To set the current default subvolume for FUP to SUBVOL1 but not change the current default volume or node:

  ```
  -VOLUME SUBVOL1
  ```

- To set the current default volume for FUP to $BOOKS3 but not change the current default subvolume or node:

  ```
  -VOLUME $BOOKS3
  ```

- To set the current default node, volume, and subvolume for FUP operations to \ITALY, $MILANO, and ARTWORK:

  ```
  -VOLUME \ITALY.$MILANO.ARTWORK
  ```

Commands Related to VOLUME

COMMAND	Function	Page
SYSTEM	Changes the FUP default node name	2-187

3 FUP Messages

This section describes the text messages that FUP might generate. Each description includes an explanation of what caused the message to be generated, explains the effect the erroneous activity has on the system process, and provides a suggested mode of recovery.

FUP generates four basic types of error messages:

- Informative messages that require no action
- Error messages (preceded by the word *ERROR*) that indicate a mistake with the command
- Warning messages (preceded by the word *WARNING*) that indicate an operation is potentially dangerous (but is executed)
- Severe-problem messages (preceded by the word *CALAMITY*)

Note. You should never receive a severe-problem message (CALAMITY). If you do, contact your service provider.

```
A PARTITION/ALTFILE NAME POINTS TO A NONEXISTENT VOLUME
```

Cause. During an attempt to create a new file (by using the CREATE or DUP command) or alter an old file, a secondary file (secondary partition or alternate-key file) was placed on a nonexistent volume.

Effect. A warning is issued, and the command execution proceeds.

Recovery. Place all alternate-key files and secondary partitions on the primary volume or on existing volumes.

```
ALL PARTITIONS OF A NON-KEY-SEQUENCED FILE MUST HAVE SAME
FORMAT VERSION
```

Cause. A FUP CREATE or SET command was used to specify a partition for a non-key-sequenced file that had a different file format than an existing partition.

Effect. The command fails.

Recovery. Reenter the command with an appropriate file format.

```
ALT FILE IS INCOMPATIBLE WITH ALT KEYS
```

Cause. Characteristics of the existing alternate-key file in the LOADALTFILE command conflict with the characteristics implied by keys defined in the primary-key file.

Effect. LOADALTFILE terminates unsuccessfully.

Recovery. Alter the ALTKEYs and ALTFILEs in the primary file.

```
ALTERNATE KEY FILE CANNOT BE AN SQL OBJECT
```

Cause. An ALTFILE option specified an SQL object as an alternate-key file.

Effect. The operation terminates.

Recovery. Retry the command by specifying a valid alternate-key file.

```
AN ALTERNATE KEY FILE HAS NO CORRESPONDING ALTERNATE KEY
```

Cause. An attempt to create or alter a file failed because an alternate-key file that had no corresponding alternate key was specified.

Effect. The CREATE or ALTER command fails.

Recovery. Retry the command after correcting the ALTKEY or ALTFILE definition.

```
AN ALTERNATE KEY GOES BEYOND THE END OF THE RECORD
```

Cause. An attempt to create or alter a file failed because the length of the alternate key (which was added to the offset of the alternate key) exceeded the record length specified for the file.

Effect. The CREATE or ALTER command fails.

Recovery. Retry the command after correcting the ALTKEY or ALTKEY definition.

```
AN ALTERNATE KEY HAS NO CORRESPONDING ALTERNATE KEY FILE
```

Cause. An attempt to create or alter a file failed. An alternate key was specified for a file, but an alternate-key file was not specified for the alternate key.

Effect. The CREATE or ALTER command fails.

Recovery. Retry the command after correcting the ALTKEY or ALTFILE definition.

```
AN ALTERNATE KEY HAS THE INVALID LENGTH OF 0
```

Cause. When entering the CREATE or ALTER command, you either omitted the KEYLEN parameter from an ALTKEY specification or specified zero for KEYLEN.

Effect. The CREATE or ALTER command fails.

Recovery. Retry the command after correcting the KEYLEN parameter.

```
ASSUMEID NOT PRESERVED AT DESTINATION
```

Cause. The ASSUMEID attribute of a file failed to transfer during a duplicate operation while using the SAVEALL option to a remote node.

Effect. The ASSUMEID attribute is not preserved.

Recovery. Use the ALTER command to change the attribute. You might have to use remote logon.

```
ATTEMPT TO RENAME A NONEXISTENT ALTFILE
```

Cause. An attempt was made in DUP *rename-options* to rename a nonexistent alternate file. The source file does not have the specified alternate-key file.

Effect. This is only a warning. Command execution proceeds, but the alternate-key file is not created.

Recovery. Check that the *rename-options* are those intended, and specify valid alternate-key file names.

```
ATTEMPT TO RENAME A NONEXISTENT PARTITION
```

Cause. An attempt was made in DUP *rename-options* to rename a nonexistent secondary partition volume. The source file does not have the specified partition.

Effect. This is only a warning. Command execution proceeds.

Recovery. Check that the values for *rename-options* are those intended, and specify valid partition file names. Retry the DUP operation if information was lost.

```
BAD ALT KEY PARAMETERS
```

Cause. The ALTKEY or the ALTFILE specifications for a CREATE or ALTER command were not correct. This situation can also occur if an existing file had invalid alternate-key parameters and an attempt was made to load one of its alternate-key files using LOADALTFILE.

Effect. The command fails.

Recovery. Reenter the command (if it is CREATE or ALTER) after correcting the alternate-key parameters. For a LOADALTFILE command, alter the existing file to contain valid alternate-key parameters.

```
BAD BLOCK LENGTH FOR VARIABLE RECORD LENGTH BLOCKING
```

Cause. An attempt to load or copy a file failed because the length of a block was either 0 or 1 byte. This error can occur if the file to be loaded or copied was filled without specifying the VAROUT option of the COPY command. Alternatively, data in the file might have been corrupted.

Effect. The COPY or LOAD command fails.

Recovery. Variable-length blocks must contain at least two bytes at the beginning of the block.

```
BAD CASE NUMBER
```

Cause. An error occurred in the program logic.

Effect. FUP ends abnormally.

Recovery. Call your service provider and describe the circumstances under which the error occurred.

```
BAD PARTITION PARAMETERS
```

Cause. The PART specification for CREATE or ALTER was wrong. For example, all partition numbers must be sequential, starting from 1. When this message appears after `FUP LOAD... PARTOF...`, it indicates that FUP found the file that should have been the primary file, but it was not a partitioned file, or it was not the primary partition for the destination.

Effect. The command fails.

Recovery. In the first case, retry the command after specifying all partitions up to the maximum partition number. In the second case, correct either the destination file name or the PARTOF volume name.

```
BAD VARIABLE RECORD LENGTH
```

Cause. An attempt to load or copy a file failed because the record length specified for a variable-length record was not valid. For example, the record length was a negative number. This error can occur if the file to be loaded or copied was filled without specifying the VAROUT option of the COPY command. Alternatively, data in the file might have been corrupted.

Effect. The COPY or LOAD command fails.

Recovery. Retry the command after specifying the VAROUT option.

```
CANNOT MIX UNIQUE AND NON-UNIQUE KEYS WITHIN AN ALTERNATE KEY
FILE
```

Cause. Unique and nonunique alternate keys cannot reside in the same alternate-key file.

Effect. The CREATE or ALTER command fails.

Recovery. Retry the command after changing the alternate-key file.

```
CAN'T CHECKSUM AUDITED FILES
```

Cause. An audited file was the object of a CHECKSUM command.

Effect. The command is not performed on the file.

Recovery. Turn off auditing if possible. This command cannot be performed on audited files.

```
CAN'T OPEN TAPE FILE FOR READ ACCESS IF DEFINE VALUE FOR USE
IS EXTEND
```

Cause. A TAPE DEFINE was specified with USE set to EXTEND, but the DEFINE name was specified as an input file in either a FUP COPY or LOAD command.

Effect. The command fails.

Recovery. Modify the DEFINE or command parameters and retry the command.

```
CAN'T OPEN TAPE FILE FOR READ ACCESS IF DEFINE VALUE FOR USE
IS OUT
```

Cause. A TAPE DEFINE was specified with USE set to OUT, but the DEFINE name was specified as an input file in either a FUP COPY or LOAD command.

Effect. The command fails.

Recovery. Modify the DEFINE or command parameters and retry the command.

```
CAN'T OPEN TAPE FILE FOR WRITE ACCESS IF DEFINE VALUE FOR USE
IS IN
```

Cause. A TAPE DEFINE was specified with USE set to IN, but the DEFINE name was specified as an output file in either a FUP COPY or BUILDKEYRECORDS command.

Effect. The command fails.

Recovery. Modify the DEFINE or command parameters and retry the command.

```
COMMAND IS NOT SUPPORTED FOR FILES ON OPTICAL DISK
```

Cause. An optical disk file was specified in a command that cannot operate on optical disk files.

Effect. The command fails.

Recovery. If you intended to specify another type of file, retry the command after correcting the file specification.

```
COMPACT OPTION IGNORED FOR NON-RELATIVE FILES
```

Cause. An attempt was made to copy or load a nonrelative file using the COMPACT option.

Effect. Execution of the command proceeds.

Recovery. Informational message; no corrective action is needed.

```
CONSISTENCY CHECK
```

Cause. A check was made of some internal structures.

Effect. The command might not finish.

Recovery. This message should never appear. If it does, contact your service provider.

```
COULD NOT FIND DEFINE.
```

Cause. A logical style DEFINE name was specified for a DEFINE class, such as class SPOOL or class TAPE, that does not exist.

Effect. The command fails.

Recovery. Specify a correct name and retry the command.

```
CRASH OPEN FLAG ON.  MUST ALTER PARTITIONS INDIVIDUALLY (I.E.
PARTONLY)
```

Cause. When an attempt was made to use FUP ALTER *filename*, NO AUDIT, FUP encountered a “file is bad” error (CRASHOPEN flag is on). *filename* was partitioned.

Effect. The command is not executed.

Recovery. Reissue the FUP ALTER command for each partition.

filename : CREATE ERR : *nnn*

Cause. A file-system error occurred while you tried a CREATE command with the indicated file.

Effect. The command fails.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

filename: DEFAULTED SUBVOLUME WILL NOT BE SUPPORTED IN THE FUTURE

Cause. A file name was entered with the volume name specified and the subvolume name unspecified. In later RVUs, you are required to specify the subvolume name when you specify the volume name in a command.

Effect. Processing continues.

Recovery. Informational message only; no corrective action is needed.

```
DEFINE DOES NOT MATCH TAPE LABEL OR TAPE LABEL IS BAD
```

Cause. An attempt to open a labeled-tape file failed with file-system error 196 (tape label record missing or incorrect).

Effect. The command fails.

Recovery. Verify that the specified DEFINE attributes match the label for the tape file. Correct, if necessary, and retry the command. If the command fails again, for file-system error 196 corrective action, see the *Guardian Procedure Errors and Messages Manual*.

```
DEFINE PROCESSING ERROR:  nnn
```

Cause. An internal error occurred while the system was processing DEFINEs.

Effect. The command fails.

Recovery. Note the error indicated by *nnn* and contact your service provider.

```
DESTINATION FILE IS NOT FROM INTERRUPTED DUP
```

Cause. A RESTART operation was attempted, but the destination file was not corrupt, which implies that the previous attempt to use DUP was successful.

Effect. The RESTART operation terminates.

Recovery. If you need to run DUP, perform an ordinary DUP operation.

```
filename : DP2 LOCK TABLES !CHANGES: STARTING AGAIN
```

Cause. The DP2 lock tables changed during the current LISTLOCK operation.

Effect. All lock information already displayed for *filename* is invalid.

FUP automatically starts over, retrieving and displaying current lock information for the file identified by *filename* in the error message.

Recovery. Informative message only; no corrective action is needed.

```
filename : DP2 LOCK TABLES CHANGING TOO FAST
```

Cause. The DP2 lock tables changed more than three times while FUP was attempting to retrieve and display lock information for the *filename* identified in the error message.

Effect. An error is generated, and the LISTLOCKS operation fails.

Recovery. Retry the LISTLOCKS operation.

```
DRIVE DOES NOT SUPPORT TAPEMODE SELECTION
```

Cause. The TAPEMODE parameter was used for a tape drive that does not support the setting.

Effect. The COPY or BUILDKEYRECORDS command continues without trying to set the tape mode.

Recovery. Informational message only; no corrective action is needed.

```
DRIVE DOES NOT SUPPORT THIS DENSITY SELECTION OR CAN ONLY
CHANGE DENSITY AT BOT
```

Cause. A DENSITYOUT parameter was specified for a tape drive that does not support density selection, or the DENSITYOUT parameter was specified when the tape was not at the beginning.

Effect. The COPY or BUILDKEYRECORDS command continues without trying to set the density.

Recovery. Informational message only; no corrective action is needed if the tape drive was at the correct density.

```
DUPLICATE SECONDARY PARTITION VOLUME NAME
```

Cause. The same volume was specified for more than one partition of a partitioned file.

Effect. The CREATE, ALTER, or DUP command fails.

Recovery. Each partition of a partitioned file must reside on a separate disk volume.

```
EDITREAD ERROR
```

Cause. An error occurred while the system was reading an EDIT file (READEDIT); it was probably a file-format error.

Effect. The command fails.

Recovery. For corrective action, see the *Guardian Procedure Errors and Messages Manual*.

```
source-file :  EMPTY RECORD FOUND AND NOT TRANSFERRED
```

Cause. A file contained empty records.

Effect. If the COMPACT option is selected in a COPY or LOAD command, this message indicates that the target file will have fewer records than the source file. The message is issued only when the first empty record is encountered.

Recovery. If you want the target file to contain the same records as the source file, retry the COPY or LOAD command with the COMPACT option deselected.

```
EMPTY SOURCE FILE
```

Cause. The source file of a LOAD command had zero records, and the EMPTYOK option was not specified. Because all existing data in the destination file of a LOAD command is purged at the beginning, this check for an empty source file ensures that a mistake in typing the LOAD command (such as interchanging the source and the destination files) does not result in the loss of data.

Effect. In interactive mode, the LOAD command terminates by returning to the FUP prompt. In noninteractive mode, the LOAD command terminates by abnormally ending.

Recovery. Check that you entered the command as intended. If the source file is empty, to achieve the effect of loading the destination with an empty file, use the EMPTYOK option.

```
$define-name $tape:  ERROR nnn
```

Cause. A Guardian file-system error was encountered on the indicated $*tape* described by $*define-name*.

Effect. The command fails.

Recovery. For appropriate corrective action for the file-system error indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
ERROR - filename: ERR 11
```

Cause. FUP could not find the primary partition on the volume given as the PARTOF parameter.

Effect. The operation fails.

Recovery. The *filename* parameter indicates the name of the primary partition that FUP required. Correct the file name and reissue the command.

```
ERROR - filename: ERR 59
```

Cause. FUP detected an error while generating statistics.

Effect. The command fails.

Recovery. Retry the command after correcting the errors. If the file was being updated concurrently, the error might be only a transient structure problem FUP encountered while reading large blocks.

```
ERROR - filename: ERR nnn
```

Cause. A file-system error occurred. *filename* indicates the name of the file that FUP was attempting to process.

Effect. The operation fails.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
EXPECTED A DISK DEVICE
```

Cause. A LISTOPENS command that specified a device other than a disk was attempted when FUP expected a disk device name.

Effect. The command fails.

Recovery. Correct the name of the LISTOPENS object and retry the command.

```
EXPECTED A NON-DISK DEVICE
```

Cause. A LISTOPENS command that specified a disk device name was attempted when FUP expected a nondisk device name.

Effect. The LISTOPENS command terminates.

Recovery. Correct the name of the LISTOPENS object and retry the command.

```
EXTENT SIZE SPECIFIED AS RECS/EXTENT FOR UNSTR. FILE
```

Cause. Corrupted data was encountered in some internal structures.

Effect. The command might not execute.

Recovery. This message should never appear. If it does, contact your service provider.

```
FILE INCOMPATIBLE
```

Cause. The source and destination files of a DUP...,OLD command are incompatible.

Effect. The attempted duplication fails.

Recovery. You cannot use the OLD option for incompatible files. However, you can purge the old file and retry the DUP command.

```
filename : FILE IS BROKEN
```

Cause. An attempt was made to duplicate a broken file.

Effect. The indicated file is duplicated, and the broken flag is set.

Recovery. Informational message only; no corrective action is needed.

```
filename : FILE IS CORRUPT
```

Cause. An attempt was made to duplicate a corrupt file.

Effect. The indicated file is duplicated, and the corrupt flag is set.

Recovery. Informational message only; no corrective action is needed.

```
FILE SYSTEM ERROR:  nnn
```

Cause. A file-system error was encountered during execution of the command.

Effect. The command fails.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
FILES CHECKSUMMED:  nn
```

Cause. CHECKSUM executed successfully. The number of files processed is indicated.

Effect. None.

Recovery. Informational message only; no corrective action is needed.

```
FILES DUPLICATED: nnn
```

Cause. *nnn* indicates the number of files from *fileset* that were duplicated.

Effect. None.

Recovery. Informational message only, indicating successful completion for all or some of the specified files.

```
FUP INTERNAL ERROR -- EXCEPTION: description
```

Cause. FUP encountered an internal error.

Effect. FUP stops.

Recovery. Write down the complete *description* and contact your service provider.

```
ILLEGAL CHARACTER
```

Cause. An illegal character was used in the attempted command or ANSI Name is entered specifying an invalid SQL identifier character.

Effect. The command fails.

Recovery. Retry the command after correcting the illegal character.

```
ILLEGAL EXTENT SIZE SPECIFIED
```

Cause. The attempted command contained an illegal extent-size specification. For example, if the extent size was specified as RECS (records-extent) for a structured file, and later computation of the extent size revealed that the extent size was too large (greater than 64 KB), this error is returned.

Effect. The command fails.

Recovery. Retry the command after correcting the extent-size values.

```
ILLEGAL FILENAME
```

Cause. The file name specified in the command was not valid.

Effect. The command fails.

Recovery. Retry the command after correcting the file name.

```
ILLEGAL KEY VALUE
```

Cause. The attempted command specified an invalid key value. For example, the FIRST option of the COPY command specified an invalid key value for position purposes.

Effect. The command fails.

Recovery. Retry the command after correcting the key value.

```
ILLEGAL OPTION
```

Cause. An option not allowed for the attempted command was used.

Effect. The command fails.

Recovery. Retry the command after correcting the option.

```
ILLEGAL QUALIFIED FILESET EXPRESSION:  specific error
```

Cause. A syntax error occurred in the qualified file-set expression. The additional text should describe the specific error.

Effect. The command fails.

Recovery. Retry the command after correcting the syntax.

```
ILLEGAL SYNTAX
```

Cause. A syntax error occurred in the attempted command.

Effect. The command fails.

Recovery. Retry the command after correcting the syntax.

```
ILLEGAL VALUE
```

Cause. The attempted command specified an illegal value: for example, SET EXT -1.

Effect. The command fails.

Recovery. Retry the command after correcting the value.

```
INDEX BLOCK SIZE SET EQUAL TO DATA BLOCK SIZE
```

Cause. An attempt was made to create a DP2 key-sequenced file with the index block size and data block size set to different values.

Effect. The CREATE command finishes, but the IBLOCK value for the file is set to its BLOCK value.

Recovery. If the modified IBLOCK value is not the value required, reset the BLOCK and IBLOCK values, and retry the command. The index block size must equal the data block size for DP2 key-sequenced files.

```
INSUFFICIENT CAPABILITY
```

Cause. An attempt was made to execute a command reserved for the super ID (255,255).

Effect. The command fails.

Recovery. Only the super ID can use the attempted command.

```
INTERNAL LOAD ERROR:  nnn
```

Cause. An internal program error occurred.

Effect. The command fails.

Recovery. Note the internal load error indicated by *nnn*, and contact your service provider.

```
INVALID SPI MESSAGE RETURNED BY SERVER
```

Cause. A RELOAD, STATUS, or SUSPEND command received an invalid SPI message from the reload server process.

Effect. The command fails.

Recovery. This is an internal error. Contact your service provider.

```
INVALID STATUS RETURNED BY SERVER
```

Cause. The reload server process returned invalid status information to a STATUS command.

Effect. The command fails.

Recovery. This is an internal error. Contact your service provider.

```
filename : KEPT
```

Cause. During execution of a DUP command with the KEEP option specified, the file indicated by *filename* was not duplicated.

Effect. None.

Recovery. Informational message only, indicating successful completion for all or some of the specified files.

```
KEY LENGTH MUST BE NONZERO
```

Cause. An attempt to create a file failed because the key length specified for the primary key was not specified or was specified as zero.

Effect. The CREATE or ALTER command fails.

Recovery. Retry the command with a legal KEYLEN value.

```
KEY SEQUENCED FILE PARTITION MUST HAVE A PARTIAL KEY
```

Cause. An attempt to create a key-sequenced partitioned file failed because a partial key was not specified for a secondary partition.

Effect. The CREATE command fails.

Recovery. Retry the command after correcting the PART definition.

```
KEYTAG NOT DEFINED FOR FILE
```

Cause. A BUILDKEYRECORDS command contained a *key-specifier* that is not defined for the file.

Effect. The BUILDKEYRECORDS command fails.

Recovery. Retry the command with key tags properly defined for the file.

```
LABELED-TAPE SERVER IS NOT AVAILABLE
```

Cause. An attempt to open a tape file failed with file-system error 195 (operation requires use of $ZSVR, but it is not running). The labeled-tape server, $ZSVR, is not running.

Effect. The command fails.

Recovery. Ask the system manager to start $ZSVR if possible and then retry the command.

```
LICENSE NOT PRESERVED AT DESTINATION
```

Cause. The transfer of a file's LICENSE attribute failed during a duplicate operation with the SAVEALL option to a remote node.

Effect. The LICENSE attribute is not preserved.

Recovery. Use the LICENSE command to license the file. Remote logon might be required.

```
LOAD ERROR:  nnn
```

Cause. An internal program error occurred.

Effect. The command fails.

Recovery. Note the load error indicated by *nnn*, and call your service provider.

```
LOCKINFO FAILED WITH ERR errnum
```

Cause. The file-system LOCKINFO procedure, which provides FUP with all lock information, returned the error *errnum*. This message appears when errors occur other than the detection of a DP2 lock table that is changing or has changed during a LISTLOCKS operation.

Effect. An error is generated, and the LISTLOCKS operation fails.

Recovery. Use the ERROR program for an explanation of *errnum*.

```
LOCKINFO NOT OBTAINED FOR PARTITIONS
```

Cause. FUP does not return lock information for partitioned files.

Effect. None.

Recovery. Informational message only; no corrective action is needed.

```
MISSING KEYWORD
```

Cause. A syntax error occurred. FUP required a keyword but found nothing or something other than a keyword.

Effect. The command fails.

Recovery. Retype the command, correcting the syntax error.

```
MISSING NUMBER
```

Cause. FUP required a number but found nothing or an alphabetic value. For example, after the SET option PART, the partition number must be provided.

Effect. The command fails.

Recovery. Retry the command after including the required number.

```
MISSING STRING
```

Cause. FUP required a string specification but did not find one. For example, for key-sequenced files, the PART specification must include the partial-key value, which is a string.

Effect. The command fails.

Recovery. Retry the command after including the required string specification.

```
MOUNT NEXT TAPE
```

Cause. This is not an error. FUP was reading from a tape file (for example, COPY, LOAD from tape), and no more records remained in the current reel, but there are more reels to read.

Effect. FUP waits for the next reel to be mounted.

Recovery. Mount the next reel.

```
MUST DUP PARTONLY IF 'NO PART[S]' SPECIFIED
```

Cause. Some internal structures were found to be corrupted.

Effect. The command might not execute.

Recovery. This message should not appear. If it does, contact your service provider.

```
MUST REARRANGE DATA AMONG PARTITIONS:  PARTONLY DISALLOWED
```

Cause. An attempt was made to duplicate a partition PARTONLY. FUP detected that the partition data would be reshuffled among the partitions.

Effect. The command fails.

Recovery. Reissue the command without the PARTONLY option, and convert the file as a whole.

```
NO CHANGE TO ALT KEY FILES
```

Cause. The OLD option of the DUP command was used.

Effect. This is only a warning that any alternate-key files must be reloaded. Duplication of the primary file proceeds.

Recovery. After the duplication completes, you need to update the alternate-key files using the LOADALTFILE command.

```
filename: NO RELOAD IN PROGRESS
```

Cause. A SUSPEND command issued against *filename* found no RELOAD operation for *filename*.

Effect. The command fails.

Recovery. Check that you spelled *filename* correctly or that *filename* exists before retrying SUSPEND.

```
filename:  NO SAFEGUARD PROTECTION
```

Cause. A Safeguard protected file was duplicated, but the new file is not Safeguard protected.

Effect. The file is duplicated; this is only a warning.

Recovery. Use the Safeguard command interpreter to establish Safeguard protection for the file.

```
filename:  NO STATUS AVAILABLE
```

Cause. No status information was found for a RELOAD operation, or the information found was invalid or inconsistent.

Effect. The STATUS command fails.

Recovery. Check that you spelled *filename* correctly or that *filename* exists before retrying STATUS.

```
NO SUCH LINE
```

Cause. The specified text for the HISTORY, !, or ? command was not found in the buffer.

Effect. None.

Recovery. Enter the correct text and try the command again.

```
NO SUCH SYSTEM
```

Cause. The attempted command specified a nonexistent node.

Effect. None.

Recovery. Reenter the command with a valid node name.

```
NOT ALL ATTRIBUTES WERE TRANSFERRED
```

Cause. The SAVEALL, SAVEID, or SOURCEDATE option of DUP was specified, and an error occurred while saving one or more of the relevant attributes.

Effect. The DUP command finishes successfully, but not all the attributes are transferred.

Recovery. Check which attributes did not transfer and transfer manually (if necessary).

```
NOT ALLOWED FOR PARTITIONED FILES
```

Cause. The attempted command was not allowed for partitioned files. For example, an attempt was made to duplicate a partitioned file, specifying the OLD option without the PARTONLY option.

Effect. The command fails.

Recovery. Check which commands are allowed with partitioned files.

```
NOT ALLOWED FOR THIS FILE TYPE
```

Cause. The source file of a BUILDKEYRECORDS command or the destination file of a LOAD command was an unstructured file.

Effect. The command fails.

Recovery. Specify a structured file in the command.

```
NOT ALLOWED HERE
```

Cause. A character was entered on the same line after the continuation character (&).

Effect. The command fails.

Recovery. Retry the command after deleting the character that follows the continuation character.

```
NOT ALLOWED IN BATCH MODE
```

Cause. An operation was attempted in batch mode that can be performed only from a terminal.

Effect. None.

Recovery. Perform the operation in interactive mode from a terminal.

```
NOT PRIMARY PARTITION
```

Cause. A secondary partition was specified in a context where secondary partitions are not allowed.

Effect. The command fails.

Recovery. Check that the file name is for the intended file, or use the PARTONLY option if applicable to the command.

```
NOTHING WAS SPECIFIED TO CHANGE
```

Cause. When an ALTER command was entered, only the PARTONLY option was specified. No changes to the file were specified.

Effect. None.

Recovery. Retry the command, specifying the required changes.

```
OLD OPTION USED:  IGNORING ANY PART, ALTFILE OR EXTENT
OPTIONS
```

Cause. An attempt was made to duplicate a file using the OLD option with either the PART, ALTFILE, or EXT option. These options are mutually exclusive.

Effect. Only the OLD option is used, and the others are ignored. This is only a warning.

Recovery. To change something using PART, ALTFILE, or EXT, do not use the OLD option.

```
ONLY ONE SOURCE FILE ALLOWED WITH RESTARTABLE OPTION
```

Cause. A RESTARTABLE DUP operation was attempted, but more than one source file was specified.

Effect. The operation terminates.

Recovery. Specify only one source file when you perform the operation again.

```
filename : OPEN ERR : nnn
```

Cause. A file-system error was encountered while attempting to open the indicated file.

Effect. The command fails.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
OPERATION NOT ALLOWED ON SQL OBJECT
```

Cause. An SQL object was specified in a command that cannot operate on these objects.

Effect. The command fails.

Recovery. If you intended to specify another type of file, retry the command after correcting the file specification. To perform operations on SQL objects, use the SQL Conversational Interface (SQLCI) for SQL/MP files and MXCI for SQL/MX files.

```
OPERATION NOT ALLOWED ON VIEW
```

Cause. A CHECKSUM command specified an SQL view.

Effect. The operation is not performed on the view but will be performed on the next object or file in the specified file set.

Recovery. If you intended to specify another type of file, retry the command after correcting the file specification.

```
OUT FILE CAN'T BE AN SQL OBJECT
```

Cause. An SQL object was specified as a list file.

Effect. The command operation terminates.

Recovery. Retry the command, specifying a valid list file.

```
filename: OUT FILE CAN'T BE ON OPTICAL DISK.
```

Cause. You specified a file on optical disk as the OUT file for the RUN FUP command (for example, RUN FUP/OUT $OPT.X.Y/) or as the OUT file for the FUP commands HELP, INFO, FILES, SUBVOLS, LISTOPENS, or SHOW (for example, HELP / OUT $OPT.X.Y/ ALL).

Effect. The command fails.

Recovery. Retry the command with an OUT file residing on magnetic disk.

```
OVERFLOW OF TABLE
```

Cause. FUP ran out of internal memory space while trying to execute the command.

Effect. FUP ends abnormally.

Recovery. Restart FUP, and retry the command. If the same error occurs, call your service provider.

```
OWNER NOT PRESERVED AT DESTINATION
```

Cause. The transfer of a file's OWNER attribute failed during a duplicate operation with the SAVEALL or SAVEID option to a remote node.

Effect. The OWNER attribute is not preserved.

Recovery. Use the GIVE command to change the owner. Remote logon might be required.

```
PARTIAL KEY VALUE FOR A PARTITION IS NOT IN ASCENDING ORDER
```

Cause. When creating or altering a partitioned file, the numbering of the partitions was not in the same order as the partial key values associated with the partitions.

Effect. The command fails.

Recovery. Retry the command after correcting the numbering of the partial key values or the partitions.

```
PARTITION ATTRIBUTES ARE INCONSISTENT
```

Cause. The file attributes of the individual partitions of a partitioned file were inconsistent with one another.

Effect. The command fails.

Recovery. Check the partitions, and ensure consistency by executing the necessary FUP ALTER commands. For all partitions, check that:

- The file type, record length, data-block length, key length, key offset, index-block length, and index and data-compression (where applicable) attributes are the same.
- For relative and entry-sequenced files, the partition extent sizes are the same as those specified when the primary partition was created.

```
PARTITION IS AN SQL OBJECT
```

Cause. A DUP operation that did not specify PARTONLY was attempted on a partitioned file, and one of the partitions is an SQL object.

Effect. None of the file is duplicated. The DUP operation continues executing on the next file in the file set.

Recovery. If you intended to specify another type of file, retry the command after correcting the file specification.

```
PRIMARY KEY TOO LONG FOR AN ALTERNATE KEY FILE
```

Cause. When you created or altered a file with a nonunique alternate key, the combined length of the primary key, the alternate key, and the 2-byte key specifier exceeded the 255-byte total allowed. Or when you created or altered a file with a unique alternate key, the combined length of the alternate key and the 2-byte key specifier exceeded the allowed 255-byte total.

Effect. The command fails.

Recovery. Retry the command, specifying values within the specified byte limits.

```
PROCESS CREATION ERROR TRYING TO CREATE SERVER:  %error
```

Cause. A RELOAD, STATUS, or SUSPEND command attempted to create a reload server, but the server could not be created.

Effect. The command fails.

Recovery. Ask the system manager to install ORSERV on $SYSTEM.SYSTEM. If this does not correct the problem, an internal error exists. In this case, contact your service provider.

```
filename : PURGE ERR : nnn
```

Cause. A file-system error occurred while the system tried to purge the indicated file.

Effect. The command fails.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
QUALIFIED FILESET ERROR
```

Cause. FUP cannot access the file specified in a qualified file set.

Effect. A file-system error (error number could be 0) is generated.

Recovery. If the error is not apparent, try using INFO on a file with a similar file set without using a qualified file set.

```
QUALIFIED FILESETS NOT PERMITTED
```

Cause. Qualified file set is either not permitted in this command, or it was used in an incorrect position.

Effect. The command fails.

Recovery. Use the qualified file set in the correct position, or retry the command without a qualified file set (if they are not permitted in this command).

```
nnn RECORDS CONTAIN INCOMPLETE ALTERNATE KEY FIELDS
(ALTERNATE KEY RECORDS NOT GENERATED)
```

Cause. In the primary file, *nnn* records had partial alternate-key fields. The actual record did not contain the full length of the alternate key. The corresponding alternate-key records were not generated in the destination file by LOADALTFILE or BUILDKEYRECORDS.

Effect. The LOADALTFILE or BUILDKEYRECORDS command finishes, but no alternate-key records are generated for the records with partial alternate keys. This is only a warning.

Recovery. Retry the command after padding the records in the primary file so that alternate keys are always completely contained in the records.

```
RECORD SIZE EXCEEDS THE APPLICABLE LIMIT.  FORMAT 2 DOES NOT
ALLOW AS BIG A RECORD AS FORMAT 1 FOR SAME BLOCK SIZE
```

Cause. A FUP command was entered that attempted to assign a record size inappropriate for the file format of the file.

Effect. The command fails.

Recovery. Reenter the command with an appropriate record size. For a list of appropriate record sizes, see Table 1-1, File Format Codes, on page 1-23.

```
RECORDS LOADED:  nn
```

Cause. A LOAD command was issued.

Effect. The operation completes, and *nn* indicates the number of records actually loaded. If the EMPTYOK option was used to load an empty source file, *nn* is 0.

Recovery. Informational message only; no corrective action is needed.

```
REELS PARAMETER NOT ALLOWED FOR LABELED TAPES
```

Cause. An attempt was made to use the REELS parameter when *in-file* for the operation was a TAPE DEFINE name with LABELS set to label processing.

Effect. The command fails.

Recovery. Use the REELS and VOLUME TAPE DEFINE attributes with multiple labeled tapes.

```
filename: RELOAD ABENDED WITH ERROR nnn
```

Cause. A RELOAD command failed before completion.

Effect. The STATUS command terminates.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
filename: RELOAD ALREADY IN PROGRESS
```

Cause. A RELOAD command was issued against *filename* while another RELOAD command was already in progress for that file.

Effect. The command fails.

Recovery. Informational message only; no corrective action is needed.

```
filename : RENAME ERR : nnn
```

Cause. A file-system error was encountered while attempting to rename the indicated file.

Effect. The command fails.

Recovery. For corrective action for the file-system error number indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
REQUIRED DELIMITER MISSING
```

Cause. FUP was expecting a delimiter but did not find it or an ANSI Name is entered without specifying a required comma, open parenthesis, or closing parenthesis.

Effect. The command fails.

Recovery. Retry the command after adding the delimiter.

```
RESTART FILE MUST HAVE FILECODE 855
```

Cause. A RESTART operation was attempted, but the restart file did not have the correct file code. The restart file was not created by FUP as part of a legitimate RESTARTABLE DUP operation.

Effect. The RESTART operation terminates.

Recovery. When you retry the operation, use a valid restart file that has file code 855 or perform an ordinary DUP operation.

```
RESTART FILE NOT ALLOWED ON OPTICAL DISK
```

Cause. For a RESTARTABLE DUP operation or a RESTART operation, the restart file was named explicitly or implicitly (that is, as a default) on an optical disk volume.

Effect. The operation terminates.

Recovery. Do not specify a restart file on an optical disk when you perform the operation again.

```
RESTART INFORMATION INVALID
```

Cause. A RESTART operation was attempted, but the restart file does not contain valid information.

Effect. The RESTART operation terminates.

Recovery. Perform an ordinary DUP operation.

```
SECURITY NOT PRESERVED AT DESTINATION
```

Cause. The transfer of a file's SECURITY attribute failed during a duplicate operation with the SAVEALL or SAVEID option to a remote node.

Effect. The SECURITY attribute is not preserved.

Recovery. Use the SECURE command to change the security. Remote logon might be required.

```
filename: SETMODE ERR nnn
```

Cause. A file-system error was encountered while attempting a SETMODE command.

Effect. The requested command fails.

Recovery. For corrective action for the file-system error indicated by *nnn*, see the *Guardian Procedure Errors and Messages Manual*.

```
SKIPIN PARAMETER NOT ALLOWED FOR LABELED TAPES
```

Cause. An attempt was made to use the SKIPIN parameter when the *in-file* for the operation was a TAPE DEFINE name with LABELS set to label processing.

Effect. The command fails.

Recovery. Use the FILESEQ TAPE DEFINE attribute to skip files on a labeled tape.

```
SKIPOUT PARAMETER NOT ALLOWED FOR LABELED TAPES
```

Cause. An attempt was made to use the SKIPOUT parameter when the *out-file* for the operation was a TAPE DEFINE name with LABELS set to label processing.

Effect. The command fails.

Recovery. Use the FILESEQ TAPE DEFINE attribute to skip files on a labeled tape.

```
{ SORTRECEIVE | SORTSEND | SORTSTART | SORTMERGEFINISH |
  SORTMERGERECEIVE | SORTMERGESEND | SORTMERGESTART          }
[ FILE ERR nn, | NEWPROCESS ERR n %nn, | INTERNAL ERR nn, ]
[ [ SORT FILE nn, ] SORT ERROR n ]
```

Cause. The indicated error or errors were reported by a SORT interface routine during execution of a LOAD, LOADALTFILE, or LISTOPENS command.

Effect. The LOAD, LOADALTFILE, or LISTOPENS command fails.

Recovery. A SORT error occurred in the specified phase. Four different types of errors are possible. FILE ERR is a standard file-system error. NEWPROCESS ERR includes the process creation error number (*n*) and additional process creation error information (*%nn*). INTERNAL ERR is a SORT internal error. Correct the error and retry the command.

```
SOURCE FILE HAS CHANGED
```

Cause. A RESTART operation was attempted, but the source file was modified since the previous attempt to use DUP.

Effect. The RESTART operation terminates.

Recovery. Perform an ordinary DUP operation.

```
SPACE ALLOCATION ERROR
```

Cause. An internal error occurred in the space allocation logic of FUP.

Effect. FUP ends abnormally.

Recovery. Restart FUP and retry the command. If the error occurs again, call your service provider.

```
SPI ERROR: error
```

Cause. The Subsystem Programmatic Interface (SPI) returned an error to a RELOAD, STATUS, or SUSPEND command.

Effect. The command fails.

Recovery. This is an internal error. Note the *error* and contact your service provider.

```
SQL RECOMPILATION REQUIRED
```

Cause. A DUP command duplicated an SQL sensitive file.

Effect. This is only a warning. The DUP operation proceeds.

Recovery. The newly created file should be SQL compiled.

```
TABLE OVERFLOW
```

Cause. Certain internal tables of FUP overflowed.

Effect. FUP ends abnormally.

Recovery. If a second attempt of the operation (after restarting FUP) fails, contact your service provider.

```
TAPE DEFINE BLOCKLEN VALUE TOO LARGE
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the BLOCKLEN value in the DEFINE was greater than 32,767 (the largest block size that FUP can accommodate).

Effect. The command fails.

Recovery. If possible, change the BLOCKLEN value to a size that FUP can accommodate.

```
TAPE DEFINE RECLEN VALUE TOO LARGE
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the RECLEN value was greater than 4096.

Effect. The command fails.

Recovery. If possible, change the RECLEN value to a size that FUP can accommodate.

```
TAPE DEFINE VALUE FOR BLOCKLEN CONFLICTS WITH BLOCKIN
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the BLOCKLEN value in the DEFINE did not equal the value of the BLOCKIN FUP parameter.

Effect. The command fails.

Recovery. Change the BLOCKLEN or BLOCKIN value and retry the command.

```
TAPE DEFINE VALUE FOR BLOCKLEN CONFLICTS WITH BLOCKOUT
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the BLOCKLEN value in the DEFINE did not equal the value of the BLOCKOUT FUP parameter.

Effect. The command fails.

Recovery. Change the BLOCKLEN or BLOCKOUT value and retry the command.

```
TAPE DEFINE VALUE FOR DENSITY CONFLICTS WITH DENSITYOUT
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the DENSITY value in the DEFINE did not equal the value of the DENSITYOUT FUP parameter.

Effect. The command fails.

Recovery. Change the DENSITY or DENSITYOUT value and retry the command.

```
TAPE DEFINE VALUE FOR EBCDIC CONFLICTS WITH EBCDICIN
```

Cause. The value of *in-file* was a TAPE DEFINE name, the EBCDIC value in the DEFINE was OFF, and the EBCDICIN FUP parameter was specified.

Effect. The command fails.

Recovery. Change the EBCDIC value in the DEFINE or omit EBCDICIN, and retry the command.

```
TAPE DEFINE VALUE FOR EBCDIC CONFLICTS WITH EBCDICOUT
```

Cause. The value of *in-file* was a TAPE DEFINE name, the EBCDIC value in the DEFINE was OFF, and the EBCDICOUT FUP parameter was specified.

Effect. The command fails.

Recovery. Change the EBCDIC value in the DEFINE or omit EBCDICOUT, and retry the command.

```
TAPE DEFINE VALUE FOR RECLEN CONFLICTS WITH RECIN
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the RECLEN value in the DEFINE did not equal the value of the RECIN FUP parameter.

Effect. The command fails.

Recovery. Change the RECIN or RECLEN value and retry the command.

```
TAPE DEFINE VALUE FOR RECLEN CONFLICTS WITH RECOUT
```

Cause. The value of *in-file* was a TAPE DEFINE name, and the RECLEN value in the DEFINE did not equal the value of the RECOUT FUP parameter.

Effect. The command fails.

Recovery. Change the RECLEN or RECOUT value and retry the command.

```
TAPE DEFINE VALUE FOR TAPEMODE CONFLICTS WITH FUP PARAMETER
```

Cause. The TAPE DEFINE attribute for TAPEMODE and the FUP parameter for TAPEMODE were both specified, and they conflict.

Effect. The COPY or BUILDKEYRECORDS command terminates.

Recovery. Modify one of the TAPEMODEs and retry the operation.

```
TAPE DENSITY AND TAPE MODE CANNOT BOTH BE SET
```

Cause. The TAPEMODE and DENSITYOUT parameters were both specified.

Effect. The BUILDKEYRECORDS command fails.

Recovery. Resubmit the command without either the TAPEMODE parameter or the DENSITYOUT parameter.

```
TAPE FULL.  MOUNT NEW SCRATCH TAPE
```

Cause. This is not an error. FUP was writing to a tape file and reached the end of the current reel.

Effect. FUP waits for the next reel to be mounted.

Recovery. Mount the next reel.

```
TAPE IS WRITE PROTECTED
```

Cause. A write request was made, but the write ring was missing.

Effect. The write request to the tape fails.

Recovery. Unmount the tape, insert a write ring, and remount the tape. Press RETURN to restart the operation.

```
TAPE MOUNT REQUEST REJECTED
```

Cause. An attempt to open a tape file failed with file-system error 194 (device use or mount request rejected by operator). The operator rejected the request to mount the specified tape.

Effect. The command fails.

Recovery. Determine why the tape mount request was rejected, resolve the problem, and retry the command. For the corrective action for file-system error 194, see the *Guardian Procedure Errors and Messages Manual*.

```
THERE IS NO SUCH ALTERNATE FILE
```

Cause. The *key-file-number* parameter to LOADALTFILE was undefined for the specified primary file.

Effect. The command fails.

Recovery. Execute a FUP INFO with the DETAIL option to see what the alternate-key files are for a given file. Use LOADALTFILE only for existing alternate-key files.

```
TOO LONG
```

Cause. A specified string was too long.

Effect. The command fails.

Recovery. Retry the command after correcting the string length.

```
TOO MANY
```

Cause. Too many objects were specified for the attempted command. For example, more than 10 file sets were specified in a file-set list for FUP, or too many key tags were specified for the BUILDKEYRECORDS command.

Effect. The command fails.

Recovery. Retry the command with an acceptable number of specified objects.

```
TOO MANY ENTRIES TO SORT, REST ARE UNSORTED
```

Cause. Too many (more than 1000) volumes were specified for the system.

Effect. Only the first 1000 volumes are sorted by name. The remaining volumes are printed in hardware device number order.

Recovery. Retry the command with 1000 or fewer volumes specified.

```
TRUNCATION OCCURRING
```

Cause. During execution of the COPY, LOAD, or LOADALTFILE command, the input records were truncated.

Effect. The command proceeds; this is only a warning.

Recovery. Check the BLOCKIN, RECIN, or input record length values. If truncation is intended, ignore the warning.

```
TRUNCATION OF LAST VARIABLE LENGTH RECORD IN BLOCK
```

Cause. This is a warning message. The VARIN option was used with the COPY or LOAD command, and the last variable-length record in the block was truncated because the record, as indicated by the record length at the beginning of the record, extended beyond the end of the block.

Effect. Execution of the command proceeds; this is only a warning.

Recovery. Check that the BLOCKIN value (if specified) is correct and that the file was generated by FUP COPY with a VAROUT option.

```
TRY THIS COMMAND ON SPECIFIC SUBDEVICES
```

Cause. The LISTOPENS command does not return information for devices that have subdevices.

Effect. You receive an error message instead of the information you requested.

Recovery. You need to execute LISTOPENS for specific terminals. An example is LISTOPENS $TC1.#C13.

```
filename: UNABLE TO SUSPEND RELOAD
```

Cause. A SUSPEND command was issued, but the RELOAD command could not be stopped.

Effect. The command fails.

Recovery. Check that you have proper security access to stop the reload server process and then retry the command.

```
UNIQUE KEYS MUST HAVE SAME LENGTH WITHIN AN ALTERNATE KEY
FILE
```

Cause. An attempt to create or alter a file failed because unique keys were not the same length within the same alternate-key file.

Effect. The command fails.

Recovery. Place unique keys with different lengths in different key files.

```
UNKNOWN COMMAND
```

Cause. The attempted command was not a FUP command.

Effect. The command fails.

Recovery. Check that what you are trying can be done using FUP. If it can, retry the command after correcting any errors in typing.

```
UNIMPLEMENTED FUNCTION
```

Cause. An attempt was made to use a function that is not available.

Effect. The request is not executed.

Recovery. This message should never appear. If it does, contact your service provider.

```
UNTERMINATED STRING
```

Cause. A quoted string was not terminated with a closing quote.

Effect. The command fails.

Recovery. Retry the command after closing the quoted string.

```
USING SPECIFIED EXTENT SIZES:  USER MUST ENSURE CONSISTENT
PARTITIONS
```

Cause. An attempt was made either to duplicate a partition, PARTONLY, by using the EXT or PART option to specify the extent sizes, or to alter the extent sizes of a secondary partition in the file label of the primary partition of a DP2 key-sequenced file.

Effect. The ALTER finishes; this is only a warning.

Recovery. Check that the extent sizes in the file label of the primary partition reflect the actual extent sizes of the secondary partitions.

```
dev name: WILL NOT RETURN OPEN INFORMATION: ERR nnn
```

Cause. The device specified in a LISTOPENS command does not return open information.

Effect. The command fails.

Recovery. For information on correcting the indicated file-system error number, see the *Guardian Procedure Errors and Messages Manual*.

```
WRITEEDIT ERROR
```

Cause. An error occurred while writing an EDIT file (WRITEEDIT); it was probably a file-format error.

Effect. The command fails.

Recovery. For corrective action, see the *Guardian Procedure Errors and Messages Manual*.

```
AN ANSI NAME KEYWORD IS EXPECTED LIKE CATALOG, SCHEMA, TABLE
OR INDEX
```

Cause. ANSI Name is entered without specifying an ANSI Name keyword.

Effect. The command fails.

Recovery. Retry the command after adding the required ANSI Name keyword.

```
ANMS FAILED WITH PROCESS_CREATE_ CALL ERROR: <error>
```

Cause. FUP returns this error message when the ANSI Names mapping service (ANMS) is unable to launch the ANSI Name Mapping Server Process (ANSP). The error code returned is the specific PROCESS_CREATE error.

Effect. The command fails.

Recovery. For corrective actions contact the SQL/MX ANSI Names developers.

```
ANMS RETURNED SQL ERROR: <error>
```

Cause. Retuned by ANSI Names Mapping Service (ANMS) to FUP when ANSI Name is entered and some miscellaneous SQL error occurs.

Effect. The command fails.

Recovery. For corrective actions, contact the SQL/MX ANSI Names developers.

```
ANMS WAS UNABLE TO PERFORM THE MAPPING BECAUSE OF FILE SYSTEM
ERROR: <error>
```

Cause. The SQL/MX ANSI Names mapping service (ANMS) is unable to perform the mapping due to a file-system error.

Effect. The command fails.

Recovery. For corrective actions contact the SQL/MX ANSI Names developers.

```
ANSI NAME IS NOT FULLY QUALIFIED
```

Cause. ANSI Name entered is not fully qualified, i.e. A standard SQL three-part name consisting of three SQL identifiers representing, respectively, a catalog name, a simple schema name, and a simple object name is not entered.

Effect. The command fails.

Recovery. Retry the command by entering the fully qualified ANSI Name.

```
BAD SQL IDENTIFIER; IDENTIFIER MUST CONTAIN AT LEAST ONE
CHARACTER
```

Cause. ANSI Name is entered with a missing SQL identifier.

Effect. The command fails.

Recovery. Retry the command after adding the required SQL identifiers.

```
BAD SQL IDENTIFIER; TOO LONG
```

Cause. ANSI Name is entered with SQL identifier that is too long.

Effect. The command fails.

Recovery. Retry the command after correcting the SQL identifier.

```
BAD SQL IDENTIFIER; UNMATCHED DOUBLE QUOTE
```

Cause. The delimited ANSI Name is entered without a matching double quote.

Effect. The command fails.

Recovery. Retry the command after adding the missing double quote.

```
COMMAND IS NOT SUPPORTED FOR ANSI NAME KEYWORD MODULE
```

Cause. ANSI Name is entered with the SQL/MX MODULE keyword.

Effect. The command fails.

Recovery. None.

```
COMMAND IS NOT SUPPORTED FOR ANSI NAME KEYWORD SCHEMA USER.
```

Cause. ANSI Name is entered with the SQL/MX SCHEMA USER keyword.

Effect. The command fails.

Recovery. None.

```
COMMAND NOT SUPPORTED FOR ANSI NAME KEYWORD CATALOG AND
SCHEMA
```

Cause. ANSI Name is entered with CATALOG or SCHEMA in the INFO or RELOAD command.

Effect. The command fails.

Recovery. None.

```
COMMAND ONLY SUPPORTS TABLE PARTITION and INDEX PARTITION
```

Cause. ANSI Name is entered with TABLE or INDEX in the RELOAD command.

Effect. The command fails.

Recovery. None.

```
DUPLICATE REQUEST, AN ANSP WAS ACTIVE
```

Cause. Returned when FUP tries to launch one more ANSI Name Server Process (ANSP) when it already has one active ANSP.

Effect. The command fails.

Recovery. This message should never appear. If it does, stop FUP and start it again and then issue the command.

```
<SQL/MX ANSI Name> DOES NOT EXIST OR IS INACCESSIBLE
```

Cause. ANSI Name entered does not exist on the system or is inaccessible.

Effect. The command fails.

Recovery. Check whether the input ANSI Name really exists in the system or not and then retry the command with the correct ANSI Name..

```
GUARDIAN FILE NAME EXPECTED
```

Cause. ANSI Name is encountered after the Guardian name in any command supporting ANSI Names or in any command that does not support ANSI Names.

Effect. The command fails.

Recovery. Retry the command by entering either the ANSI Name or the Guardian name.

```
INTERNAL ERROR OCCURED IN ANSP DUE TO MEMORY ALLOCATION
FAILURE
```

Cause. Returned by FUP when an internal error occurs in the ANSI Name Server Process (ANSP) due to a memory allocation failure.

Effect. The command fails.

Recovery. For corrective actions, contact the SQL/MX ANSI Names developers.

```
REQUIRED SINGLE QUOTE MISSING; ANSI NAME EXPECTED
```

Cause. ANSI Name is entered without specifying a required starting single quote or if guardian file name is entered after the first ANSI Name.

Effect. The command fails.

Recovery. Retry the command after adding the required missing single quote. If you have entered the Guardian name along with the ANSI Name in the same command, then retry the command by entering either the ANSI Name or the Guardian name.

```
REQUIRED SINGLE QUOTE MISSING OR BAD ANSI NAME ENTERED
```

Cause. ANSI Name is entered without specifying a required ending single quote or the ANSI Name is not entered in the proper format.

Effect. The command fails.

Recovery. Retry the command after adding the required missing single quote or by correcting the ANSI Name.

```
SPACE CHARACTER MISSING
```

Effect. ANSI Name is entered with space character missing between the ANSI Name keyword and the SQL identifier.

Effect. The command fails.

Recovery. Retry the command after adding the required space character.

```
SQL INDENTIFIER MISSING
```

Cause. ANSI Name is entered without specifying a required SQL identifier.

Effect. The command fails.

Recovery. Retry the command after adding the required SQL identifier.

```
SQL/MX IS NOT INSTALLED ON THE SYSTEM
```

Cause. ANSI Name is entered but SQL/MX is not installed on the system.

Effect. The command fails.

Recovery. Install SQL/MX on the system.

```
SQL/MX ANSI NAMES NOT SUPPORTED
```

Cause. Returned by FUP when SQL/MX ANSI Names APIs are not installed on the system.

Effect. The command fails.

Recovery. Install the SQL/MX ANSI Names SPR on the system.

```
VERSION MISMATCH BETWEEN SQL/MX MAPPING SOFTWARE AND FUP
```

Cause. There is a version mismatch between FUP and the SQL/MX MAPPING SOFTWARE (SQL/MX ANSI Names mapping service - ANMS).

Effect. The command fails.

Recovery. Install the version of FUP and the SQL/MX MAPPING SOFTWARE (SQL/MX ANSI Names mapping service ANMS) that are compatible with each other.

```
VERSION MISMATCH BETWEEN SQL/MX PARSER AND FUP
```

Cause. There is a version mismatch between FUP and the SQL/MX parser.

Effect. The command fails.

Recovery. Install the version of FUP and SQL/MX parser that are compatible with each other.

A DEFINE Tables

You can use SPOOL, TAPE, or MAP DEFINEs to specify information for a FUP process before you start it:

- Use SPOOL DEFINEs to send command output to a spooler.
- Use TAPE DEFINEs to send command output to a tape file or receive a tape file as input.
- Use MAP DEFINEs to substitute a logical name for an actual file name.

Note. Do not use TAPE DEFINE attributes that conflict with your FUP command parameters. For more information about DEFINEs, see the *Guardian User's Guide .*

Use these tables to help specify the options you need.

Table A-1. How FUP Input Options Work With TAPE DEFINEs

FUP Parameter	FUP Parameter Only	FUP Parameter and DEFINE	DEFINE Only	DEFINE
BLOCKIN	BLOCKIN is the size of the buffer used by FUP	FUP quits with an error if values conflict	BLOCKIN is set to BLOCKLEN	BLOCKLEN
EBCDICIN or XLATE	FUP performs translation	FUP performs translation and sets DEFINE OFF, or an error occurs if values conflict	FUP lets label processing do the translation	EBCDIC
RECIN	RECIN used for FUP deblocking	FUP quits with an error if values conflict	RECIN is set to RECLEN	RECLEN
REELS	Error	Error	FUP ignores DEFINE	REELS
REWINDIN	DEFINEs do not affect this parameter			N.A.
SKIPIN	Error	Error	FUP ignores DEFINE	FILESEQ
UNLOADIN	DEFINEs do not affect this parameter			N.A.

Table A-2. How FUP Output Options Work With TAPE DEFINEs

FUP Parameter	FUP Parameter Only	FUP Parameter and DEFINE	DEFINE Only	DEFINE
BLOCKOUT	BLOCKLEN is set to BLOCKOUT	FUP quits with an error if values conflict	BLOCKOUT is set to BLOCKLEN	BLOCKLEN
DENSITYOUT	FUP sets DEFINE to DENSITYOUT and sets density	FUP sets density, or an error occurs if values conflict	FUP ignores the DEFINE	DENSITY
EBCDICOUT or XLATE	FUP performs translation	FUP performs translation and sets DEFINE OFF, or an error occurs if values conflict	FUP lets label processing do the translation	EBCDIC
RECOUT	RECLEN is set to RECOUT and is used by FUP for blocking	FUP quits with an error if values conflict	RECOUT is set to RECLEN, and it is then used by FUP for blocking	RECLEN
REWINDOUT	DEFINEs do not affect this parameter			N.A.
SKIPOUT	Error	Error	FUP ignores DEFINE	FILESEQ
UNLOADOUT	DEFINEs do not affect this parameter			N.A.

B FUP Command Summary

For a description of the individual function and structure of each FUP command, see Section 2, FUP Commands. Each command fits into one of four distinct command groups: control, information, security, and file management. The FUP commands in this appendix are categorized according to their function.

Table B-1 lists the commands you use to control FUP. Table B-2 lists the commands you use to obtain information.

Table B-3 lists commands used for security management. You can use these commands only on files with standard security codes. For files protected by the Safeguard product, you must use the Safeguard command interpreter (SAFECOM).

Note. For information on SAFECOM commands and their syntax, see the *Safeguard Reference Manual*.

Table B-4 lists commands that you can use to manage files with either Guardian or Safeguard security codes.

Table B-1. FUP Control Commands

Command	Function
!	Reexecutes the specified command.
?	Displays the specified command.
ALLOW	Sets the number of errors and warnings allowed during the execution of FUP commands.
CONFIG[URE]	Customizes your FUP configuration information.
DISPLAYBITS	Sets the display mode for file data.
EXIT	Terminates the FUP process. It must be typed at the FUP prompt.
FC	Lets you fix a command by editing and reexecuting a command line. This command is for interactive use only.
HELP	Lists the names of all the FUP commands or displays the syntax of a particular command.
HISTORY	Displays the FUP commands used most recently.
OBEY	Reads commands from a file and executes them.
REPORTWIDTH	Sets the maximum length (in columns) for the FUP output format.
SYSTEM	Sets the current default system name used by FUP.
VOLUME	Sets the current default disk volume and subvolume names used by FUP.

Table B-2. FUP Informational Commands

Command	Function
FILENAMES	Displays the names of files.
FILES	Displays the names of all files in a given subvolume or volume.
INFO	Displays the file characteristics of one or more files.
LISTLOCKS	Displays information on all locks (granted or waiting) for a specified file set.
LISTOPENS	Lists all processes that currently have one or more designated files open.
STATUS	Displays information about the progress of a RELOAD operation.
SUBVOLS	Displays the names of all subvolumes on a designated volume.
VOLS	Displays information about volumes on a system.

Table B-3. FUP Security Management Commands

Command	Function
GIVE	Changes a file owner ID for one or more files. For Safeguard protected files, you can change ownership only through the Safeguard command interpreter (SAFECOM).
LICENSE	(Super-ID command) Licenses one or more program files containing privileged procedures so that non-privileged users can run the files. For Safeguard protected files, you must perform this function through SAFECOM.
REVOKE	(Super-ID command) Revokes the license of one or more program files to execute with privileged procedures. For Safeguard protected files, you must perform this function through SAFECOM. When the CLEARONPURGE or PROGID option is included in the REVOKE command, the appropriate attribute (CLEARONPURGE or PROGID) is revoked. For Safeguard protected files, you must perform this function through SAFECOM.
SECURE	Sets file security attributes for one or more disk files. For Safeguard protected files, you must perform this function through SAFECOM.

Table B-4. FUP File Management Commands (page 1 of 2)

Command	Function
ALLOCATE	Preallocates a specified number of file extents for one or more disk files.
ALTER	Changes selected characteristics of a disk file.
BUILDKEYRECORDS	Writes the alternate-key records for key fields of a specified structured disk file to a destination file (usually a magnetic tape). You can then load the alternate-key records from the destination file into the alternate-key file with the COPY or LOAD commands.
CHECKSUM	Recomputes the checksum value for each block of data in a file.

Table B-4. FUP File Management Commands (page 2 of 2)

Command	Function
COPY	Makes record-by-record copies of files to and from the same or different media. It can also display the contents of a file.
CREATE	Creates a file using the current file-creation parameter values that have been defined with a SET command.
DEALLOCATE	Deallocates any extents past the end-of-file extent for one or more disk files.
DUP[LICATE]	Makes a copy of one or more disk files. There are special considerations for duplicating Safeguard protected files. Refer to the syntax and guidelines for this command.
LOAD	Loads data into a structured disk file without affecting any associated alternate-key files. For key-sequenced files, the input data can be unsorted or sorted. Unless you specify sorted, the LOAD command assumes that data is unsorted and sorts the input records before loading the file. For key-sequenced files, you can also specify slack space for future insertions to the file.
LOADALTFILE	Loads an alternate-key file with the alternate-key records of a specified structured disk file. You can specify slack space for future insertions.
PURGE	Purges one or more disk files.
PURGEDATA	Purges data from one or more disk files.
RELOAD	Reorganizes a key-sequenced file while permitting full access to it.
RELOCATE	Moves files on SMF virtual disks from one physical volume to another (within a storage pool).
RENAME	Renames one or more disk files.
RESET	Changes one or more file-creation parameter values to the default settings.
RESTART	Restarts a RESTARTABLE DUP operation.
SET	Sets one or more file-creation parameter values for subsequent file creations. To set Safeguard parameter values, you must use SAFECOM instead of FUP.
SHOW	Displays the current settings of the file-creation parameter values.
SUSPEND	Temporarily stops a RELOAD operation.

C

FUP Command Syntax Summary

To run FUP:

```
FUP [ / run-options / ] [ command ]
```

To specify files, you can use *fileset* or *fileset-list*.

```
   fileset is:
   [[[ \node.]$volume.]subvolume.| *.]{file-id | *}
   fileset-list is:
   { fileset | ( fileset [ , fileset ] ... ) }
! [ -num | num | string | "quoted" ]
? [ -num | num | string | "quoted" ]
```

```
ALLOCATE fileset-list , num-extents [ , PARTONLY ]
```

```
ALTER filename { , alter-option }...
```

alter-option for all file types is:

```
[ NO ] AUDIT
[ NO ] AUDITCOMPRESS
[ NO ] BUFFERED
BUFFERSIZE unstructured-buffer-size
CODE file-code
LOCKLENGTH genric-lock-key-length
MAXEXTENTS maximum-extents
NOPURGEUNTIL timestamp
[ NO ] REFRESH
RESETBROKEN
RESETCORRUPT
[ NO ] SERIALWRITES
[ NO ] VERIFIEDWRITES
```

alter-option for files with alternate-key fields is:

```
ALTFILE ( key-file-number , filename )
ALTKEY ( key-specifier { , altkey-param }... )
DELALTFILE key-file-number
DELALTKEY key-specifier
```

alter-option for partitioned files is:

```
PART ( sec-partition-num ,
     [ \node.]$volume
     [ , pri-extent-size [ , sec-extent-size ]] )
PARTONLY
```

alter-option for odd unstructured files is:

```
ODDUNSTR
```

```
BUILDKEYRECORDS primary-filename , out-filename
              , key-specifier-list [ , out-option ]

   ...key-specifier-list is:

   { key-specifier                                 }
   { ( key-specifier [ , key-specifier ] ... ) }

   out-option is:

   BLOCKOUT out-block-length
   DENSITYOUT density
   EBCDICOUT
   PAD [ pad-character ]
   RECOUT out-record-length
   [ NO ] REWINDOUT
   SKIPOUT num-eofs
   TAPEMODE mode
   [NO] UNLOADOUT
   XLATE translation-table-name
   XLATEIN translation-table-name
   XLATEOUT translation-table-name
```

```
CHECKSUM fileset-list [ , PARTONLY ]
```

```
CONFIG[URE] config-command [ config-params ]...

   config-command is:

   command-option|environment-option

   command-option is:

   COPY copy-option [, copy-option ]...
   DUP[LICATE] dup-option [, dup-option ]...
   LOAD load-option [, load-option ]...
   LOADALTFILE loadaltfile-option [, loadaltfile-option
   ]...
   RELOAD reload-option [, reload-option ]...

   environment-option is:

   ALLOW allow-option [, allow-option ]...
   DISPLAYBITS bitcount
   [ NO ] ECHO [ CONFIG[URE] ] [ OBEY ]
   IOTIMEOUT time
   NETBLOCKSIZE size
   [ NO ] PROMPT [ PURGE ]
   REPORTWIDTH width
   RESTARTUPDATE time
   XLATE xlate-table-name [ TEXT|CHARMAP ]
                              [ IN filename  ]
```

```
COPY in-filename [ ,[ out-filename ] [,copy-option ] ... ]

   copy-option is:
   control-option
   out-option
   display-option

      control-option is:
      COUNT num-records
      FIRST { ordinal-record-num        }
         { KEY { record-spec | key-value }  }
         { key-specifier ALTKEY key-value  }
      UNSTRUCTURED
      UPSHIFT

      in-option is:
      BLOCKIN in-block-length
      [ NO ] COMPACT
      EBCDICIN
      RECIN in-record-length
      REELS num-reels
      [ NO ] REWINDIN
      SHARE
      SKIPIN num-eofs
      TRIM [ trim-character ]
      [ NO ] UNLOADIN
      VARIN

      out-option is:
      BLOCKOUT out-block-length
      DENSITYOUT density
      EBCDICOUT
      FOLD
      PAD [ pad-character ]
      RECOUT out-record-length
      [ NO ] REWINDOUT
      SKIPOUT num-eofs
      TAPEMODE mode
      [ NO ] UNLOADOUT
      VAROUT
      XLATE translation-table-name
      XLATEIN [ translation-table-name ]
      XLATEOUT [ translation-table-name ]
```

```
COPY continued ...

      display-option is:
      O[CTAL]
      D[ECIMAL]
      H[EX]
      BYTE
      A[SCII]
      NO HEAD
      [ NO ] TITLE
```

```
CREATE filename [ , create-param ] ...

   (For a description of create-param, see the SET command.)
```

```
DEALLOCATE fileset-list [ , PARTONLY ]
```

```
DUP[LICATE] from-fileset-list , to-fileset
    [ , RESTARTABLE [ restart-filename ] }
    [ , rename-option ] ...
    [ , EXT [ extent-size ]
    [       ( pri-extent-size , sec-extent-size ) ]
    [ , KEEP |, NEW |, OLD |, PURGE ]
    [ , PARTONLY ]
    [ , SAVALL |, SAVEID |, SOURCEDATE ]
    [ , PHYSVOL [ physvol ] ]


   rename-option is:

   [ ALTFILE ( key-file-number , filename )              ]
   [ PART ( sec-partition-number , [ \node.]$volume    ]
   [       [ , pri-extent-size [ , sec-extent-size ] ] ) ]
```

```
EXIT
```

```
FC [ -num | num | string | quoted ]
```

```
FILENAMES [ / OUT listfile / ] [ fileset ]
```

```
FILES [ / OUT listfile / ] [ subvolset ]

   subvolset is:

   [[ \node.][[$volume.]{ subvolume | * }]]
```

```
GIVE fileset-list , { groupnum , usernum |
groupname.username }
     [ , PARTONLY ]
```

```
HELP [ / OUT listfile / ] [ command | ALL [, SYNTAX ] |
NEWS ]
```

```
HISTORY [ / OUT listfile / ] [ num
```

```
INFO [ / OUT listfile / ] [ fileset-list / ansiname-list ]

     [ , DETAIL                                ]
     [ , EXTENTS                               ]
     [ , STAT[ISTICS] [ , PARTONLY ] [ , PARTIAL num ] ]
     [ , USER { groupnum , usernum }   ]
     [        { groupname.username }   ]
```

```
LICENSE fileset-list
```

```
LISTLOCKS [ / OUT listfile / ] fileset-list / ansiname-
list
          [ , GRANTED ] [ , DETAIL ] [ , PARTONLY ]
```

```
LISTOPENS [ / OUT listfile / ] fileset-list / ansiname-
list
         [ , SCRATCH scratch-filename ]
```

```
LOAD in-filename , destination-filename [, load-option]...

   load-option is:

   EMPTYOK
   FIRST
   PAD [ pad-character ]
   in-option
   key-seq-option

      in-option is:

      BLOCKIN in-block-length
      [ NO ] COMPACT
      EBCDICIN
      RECIN in-record-length
      REELS num-reels
      [ NO ] REWINDIN
      SHARE
      SKIPIN num-eofs
      TRIM [ trim-character ]
      [ NO ] UNLOADIN
      VARIN
      XLATE [ translation-table-name ]
      XLATEIN [ translation-table-name ]
      XLATEOUT [ translation-table-name ]

      key-seq-option is:

      MAX num-records
      PARTOF $volume
      SCRATCH scratch-filename
      SORTED
      DSLACK percentage
      ISLACK percentage
      SLACK percentage
```

```
LOADALTFILE key-file-number , primary-filename
            [ , key-seq-option ] ...

   key-seq-option is:

   MAX num-records
   SCRATCH scratch-filename
   DSLACK percentage
   ISLACK percentage
   SLACK percentage
```

```
OBEY filename
```

```
PURGE [ ! ] fileset-list [ , [ NO ] LISTALL ] [ ! ]
```

```
PURGEDATA fileset-list [ , PARTONLY ]
```

```
RELOAD [ / OUT listfile / ] filename / 'ansiname'
   [ [ NO ] DEALLOCATE ]
   [ , NEW             ]
   [ , PARTOF $volume  ]
   [ , RATE percentage ]
   [ DSLACK percentage ]
   [ ISLACK percentage ]
   [ SLACK percentage  ]
   [ RECLAIM           ]
```

```
RELOCATE logical-set [ ,physvol ] [ , [ NO ] MIRRORED ]
```

```
RENAME old-fileset-list , new-fileset [ , PARTONLY ]
```

```
REPORTWIDTH width
```

```
RESET [ create-spec [ , create-spec ] ... ]
RESET [ reset-opts | CONFIG[URE] config-simple-opts ]
```

For a description of *reset-opts* (*create-spec*), see the SET command.
For a description of *config-simple-opts*, see the CONFIG[URE] command.

```
RESTART [ restart-filename ]
```

```
REVOKE fileset-list [ , secure-option ] ...
```

secure-option is:

```
CLEARONPURGE
PARTONLY
PROGID
```

```
SECURE fileset-list [ , [ security ] [ , secure-option
]...
```

security is:

```
      { [ " ] security-string  [ " ] }
      { security-num                 }
```

secure-option is:

```
{ CLEARONPURGE | PARTONLY | PROGID }
```

```
SET create-param [ , create-param ] ...
```

create-param for all file types is:

```
[ NO ] AUDIT
CODE file-code
EXT { extent-size                          }
    { ( pri-extent-size , sec-extent-size ) }
FORMAT formatcode
LIKE filename
[ NO ] REFRESH
TYPE file-type
```

create-param for all structured files is:

```
BLOCK data-block-length
REC record-length
```

create-param for key-sequenced files is:

```
[ NO ] COMPRESS
[ NO ] DCOMPRESS
[ NO ] ICOMPRESS
KEYLEN key-length
KEYOFF key-offset
```

create-param for partitioned files is:

```
PART ( sec-partition-num , [\node ]$volume
        [ , pri-extent-size [ , [ sec-extent-size ]
           [ , partial-key-value ] ] ] )
[ NO ] PARTONLY
```

(continued from previous page)

create-param for files with alternate-key fields is:

```
[ NO ] ALTCREATE
ALTFILE ( key-file-number , filename )
ALTKEY ( key-specifier { , altkey-param }... )
```

where *altkey-param* is any one of:

```
FILE key-file-number
[ NO ] INSERTIONORDER
KEYLEN key-length
KEYOFF key-offset
NO NULL
NULL null-value
[ NO ] UNIQUE
[ NO ] UPDATE
```

create-param for DP2 files is:

```
[ NO ] AUDITCOMPRESS
[ NO ] BUFFERED
MAXEXTENTS maximum-extents
QUEUEFILE
[ NO ] SERIALWRITES
[ NO ] VERIFIEDWRITES
```

create-param for odd unstructured files is:

```
BUFFERSIZE unstructured-buffer-size
ODDUNSTR
```

create-param for files on SMF virtual disks is:

```
PHYSVOL physvol
```

```
SHOW [ / OUT listfile / ]
     [ show-opts                                              ]
     [ CONFIG[URE] [ AS COMMANDS ] [ config-simple-opts ]]
```

For a description of *config-simple-opts*, see the CONFIG[URE] command.

(continued from previous page)

show-opts is:

```
ALTCREATE
ALTFILE [ key-file-number ]
ALTFILES
ALTKEY [ key-specifier ]
ALTKEYS
AUDIT
AUDITCOMPRESS
BLOCK
BUFFERED
BUFFERSIZE
CODE
COMPRESS
DCOMPRESS
EXT
FORMAT
ICOMPRESS
KEYLEN
KEYOFF
MAXEXTENTS
ODDUNSTR
PARTONLY
PART [ partition-num ]
PARTS
REC
REFRESH
SERIALWRITES
TYPE
VERIFIEDWRITES
```

```
STATUS [ / OUT listfile / ] filename [ , DETAIL ]
```

```
SUBVOLS [ / OUT listfile / ] [ subvolset ]
```

subvolset is:

```
[ \node.] $volume [.subvol ] | subvol
```

```
SUSPEND [ / OUT listfile / ] filename
```

```
SYSTEM [ \node [.$volume ] [.subvolume ] ]
```

```
VOLS  [ / OUT listfile / ] [ volset ]
```

volset is:

```
[ \node.] $volume
```

```
VOLUME [ [\node.]$volume [.subvol ] ]
       [ subvol  ]

FILESET LIST { fileset | ( fileset [ , fileset ] ... ) }

FILESET filename [ qualexpr ]
```

where *filename* can have wild-card characters.

```
SUBVOLSET [ \node.] $volume [.subvol ] | subvol

VOLSET [ \node.] $volume

FILELIST filename [ , filename ] ...

FILENAME [[[ \node.[ $volume.[ subvol. ] file-id

VOLNAME [ \node.] $volume

QUALEXPR qualifier
```

qualifier is:

```
EXCLUDE fileset-list
FROM CATALOG[S] catalog-list
START file-id
WHERE expression
```

catalog-list is:

```
[ \node.] $volume [.subvol ] | subvol
```

expression is:

```
( expression )
expression AND expression
expression OR expression
NOT expression

file-attribute
OWNER userid
timestamp-field time-conditional time-value

FILECODE conditional-number

EOF conditional-number
```

file-attribute is:

```
    ALTKEY
    AUDITED
    BROKEN
    COLLATION
    CORRUPT
    CRASHOPEN
    ENSCRIBE
```

```
      ENTRYSEQUENCED
      FORMAT1|FORMAT2
      INDEX
      [ SHORTHAND | PROTECTION ]  VIEW
      KEYSEQUENCED
      LICENSED
      OPEN
      PROGID
      RELATIVE
      [ PRIMARY | SECONDARY ]  PARTITION
      ROLLFORWARDNEEDED
      SAFEGUARD
      SQL
      SQLPROGRAM
      TABLE
      TRUSTED
      TRUSTME
      TRUSTSHARED
      UNSTRUCTURED
```

user-id is:

```
group-name.user-name
group-name.*
group-number, user-number
group-number,*
```

timestamp-field is:

```
CREATIONTIME
EXPIRATIONTIME
LASTOPENTIME
MODTIME
```

time-conditional is:

```
AFTER  | >
BEFORE | <
```

time-value is:

```
[ date ] time
[ time ] date
```

time is:

```
hh:mm [:ss]
```

date is:

```
dd mmm yyyy
mmm dd yyyy
```

Glossary

abend. The abnormal end of a task or process, or an error condition that can result in the termination of a program. The FUP process terminates with an abend message if it encounters an error in a command.

access mode. A specification that determines the different types of operations (read-write, read-only, or write-only) that a process can perform against a file. The mode is established when the file is opened.

alternate key. A data field that provides an alternative access path to the records in a file. This path differs from the inherent access path (the primary key) defined for the file. Unlike primary key values, alternate key values need not be unique. An example is an employee data file. An alternate key might be used for the department number of the employee. Using this key, a program could access the data records for all employees who work in a particular department.

American National Standards Institute (ANSI). An organization affiliated with the International Organization for Standardization (ISO). It establishes procedures by which voluntary industry standards are created and maintained. Accepted and proposed standards include transmission code and protocol (ASCII), media (tape and diskette), and languages (FORTRAN and COBOL).

American Standard Code for Information Interchange (ASCII). A method of coding data that consists of seven bits for each character (plus a parity bit). Designed for synchronous or asynchronous use, the code has 128 standard characters. It can also be an 8-bit code representing characters and control codes. One of two major codes used to represent and exchange data. (The other is EBCDIC.)

auditing. The monitoring of transactions in preparation for recovery efforts.

backup. The hardware and software resources available to recover after a degradation or failure of one or more system components.

binary. A numbering system based on two's rather than 10's. The binary system uses only the digits 0 and 1.

binary digit (bit). In the binary notation, either of the characters 0 or 1. Smallest unit of information used in data processing. Eight bits equal one byte. A single unit of information in a circuit.

block. A data collection that can be read from or written to storage media, or read and written in a single operation. This grouping increases the efficiency with which the media are used.

buffer. One of three things: (1) a storage area for a block of data, (2) a storage device used to compensate for a difference in the rate of data flow when transmitting data from one

device to another, or (3) a temporary or dynamic storage facility. It carries both logical (use within a program) and physical connotations.

byte. An 8-bit storage unit; the smallest addressable unit of memory representing one alphanumeric character.

cache. A portion of memory that temporarily stores an entire directory, a set of names, or a set of files.

catalog. A collection of files that maintains context for virtual disk processes, pools, or $ZSMS.

checksum. A technique for checking the validity of a multiple-byte block of data by arithmetically adding all the bytes together to obtain a sum. If the sum does not match the one that was previously computed, an error exists somewhere within the block of data.

COBOL. The compiler and run-time support for the American National Standard Programming Language Common Business-Oriented Language (COBOL). The code for most Pathway server processes is written in this language.

DEFINE attribute. A name identifying a class of DEFINE (TAPE or SPOOL) or a particular object (FILE or SUBVOL). A DEFINE attribute is always associated with a value as an attribute/value pair in a named DEFINE.

A named set of attributes and associated values. A user can specify information that jobs communicate to processes they start. Default DEFINEs hold the standard default values of a process such as the default volume. Spool DEFINEs pass information to the spooler collector process. The attributes of a spool DEFINE specify parameters such as the spooler location and batch name. SQL catalog DEFINEs specify the locations of SQL/MP catalogs. A user can enter the logical name of the catalog DEFINE in place of a catalog name in CATALOG clauses in SQL/MP data manipulation language (DML) statements. Tape DEFINEs pass information to the tape process during labeled-tape operations. Tape DEFINE attributes specify parameters such as the tape device name and the record format.

direct file. A file that is not on an SMF virtual disk; its logical (external) name and physical (internal) file are identical. All files on direct volumes are direct, and files on physical volumes in pools can also be direct.

direct volume. A volume that is not in any pool. All files contained in a direct volume are direct files.

Disk Compression Program (DCOM). A software product for NonStop systems that compresses or joins together the pieces of fragmented disk files to increase usable disk space.

Disk Space Analysis Program (DSAP). A software product for NonStop systems that is used to analyze use of space on a disk.

Disk Process 2 (DP2). The portion of the operating system software that performs read, write, and lock operations on disk volumes. The disk process also implements Enscribe and SQL/MP file types; creates TMF audit-trail records; performs logical REDO operations for Remote Duplicate Database Facility (RDF); and manages disk space, disk controllers, and paths to the disks. This disk process provides enhanced performance, throughput, recoverability, and reliability improvements in high-volume, online transaction processing situations.

EBCDIC. An 8-bit coded character set. One of two major codes used to represent and exchange data. (The other is ASCII.) EBCDIC uses the eighth bit for an information bit, which extends the range of distinct characters to 256.

EDIT file. An unstructured file containing text that can be processed by either of the editors for NonStop systems—EDIT or PS Text Edit (TEDIT). An edit file typically contains either source program code or documentation.

end-of-file (EOF) marker. A marker placed at the end of a file to indicate that the file contains no additional information.

Enscribe. The database file-management software provided as part of the Guardian file system. The Enscribe software provides access to—and manipulation of—records in a database on a NonStop system. Files on a Guardian system can be either Enscribe files or SQL/MP tables.

entry-sequenced file. A file in which each new record is stored at the end of the file in chronological sequence and whose primary key is the system-generated record address. Records (rows) can be updated but not deleted.

exclusion mode. A specification that determines the degree of access that competing processes can have to a file simultaneously (shared access, exclusive access, or protected access). Exclusion mode is specified when the file is opened.

extent. Either a contiguous area on disk for allocating one file, or a block of physical storage for allocating files.

extent sizes. The size in bytes of a contiguous area on disk for allocating one file.

external name. The file name known and used by applications. For logically named SMF files, it is the logical name. For direct files, it is the direct name.

file partitioning. The action of subdividing a large file into smaller units that can be handled more effectively.

file set. A file or a set of files that can be specified (optionally) with the wild-card option to define name patterns. The file or file set can usually be qualified with an additional set of exclusions.

Format 1 file. A Format 1 file is a file created on RVUs preceding D46.00 or G06.00.

Format 2 file. Beginning with the D46.00 and G06.00 RVUs, a Format 2 file can contain larger blocks and partitions than a Format 1 file.

internal file name. The name by which a file is known to DP2. For a direct file, it is the same as the file name. For a file that is name-managed by SMF, the physical file name is on a subvolume beginning with ZYS (for logically named files) or ZYT (for logical templates). See also physical file name.

key-sequenced files. A structured Enscribe file consisting of variable-length records that are accessed by the values contained within designated key fields. The records in a key-sequenced user database are stored logically in ascending order, according to the value contained in their primary key field.

keyword. A reserved word with a fixed meaning in a program. It cannot be changed by the user.

LIN. See location-independent naming (LIN).

location-independent naming (LIN). The naming of files so that their names do not determine their physical location.

logical file name. A file name whose volume component is an SMF virtual disk. See also external name.

logical temporary file. A temporary file (the file name beginning with "#") created on a logical volume. The volume component of its file name is that logical volume.

logically-named file. A file whose name is a logical file name.

node. A uniquely identified system location on a network. Guardian files are stored at a particular node, in a specific disk volume and subvolume. The node name is preceded by a backslash (\); for example, \WEST.

nonaudited file. A disk file that is not flagged for auditing by the HP NonStop Transaction Management Facility (TMF).

null value. A value indicating that a program has made no explicit assignment to a variable or field. In the context of the SPI and DSM programmatic interfaces, a field of a structure has a null value if the application has made no explicit assignment to that field after calling the SSNULL procedure to initialize the structure.

pad character. A character inserted as fill when insufficient data characters are present to satisfy a length requirement.

parameter. A name specified in a prepared command for which the user substitutes a value when executing the command.

partition. The portion of a partitioned table, index, or physical file that resides on a particular disk volume.

physical disk process. A DP2 process.

physical file name. The internal name by which a file is known to DP2. For a direct file, this name is the same as the file name. For a file name-managed by SMF, the physical file name is on a subvolume beginning with ZYS (for logically named files) or ZYT (for logical templates). See also internal file name.

physical volume. A disk volume managed by DP2.

physical volume process. A DP2 process.

PHYSVOL option. A physical volume name optionally specified when a logical file is created (either programmatically or through a command), instructing SMF to create the file on that volume. Examples include CREATE interfaces and FUP RELOCATE.

pool. A collection of physical volumes managed by SMF. Also known as storage pool.

pool process. The SMF process managing a storage pool, with the same name as that storage pool.

process identification number (PIN). A number assigned to a process for identification purposes.

qualified file-set list. A list of file sets qualified by a clause (optionally) that selects objects based on characteristics of the file. To determine which qualifiers are permitted, refer to the syntax.

record. A basic unit of storage in a file or database pertaining to a particular item. A record is the smallest logical unit of data that can be read from or written to a file. A record in a relational table is represented as a row. A record can also be any audit record that describes the outcome of a transaction. More specifically, a status record is either a commit record or an abort record. A record can also be a Data Definition Language (DDL) dictionary object that describes the structure of an Enscribe disk file. A record usually includes file-creation information, which allows the File Utility Program (FUP) to create a file from the record structure. If the file is to be key sequenced, a record also contains the key attributes.

relative files. A file in which each new record is stored at the relative record location specified by its primary key, and whose primary key is either a user-defined or system-defined relative record number that indicates the record location. Each record location can be regarded as a numbered slot that holds one fixed-length record, with the primary key identifying the slot number. In general use, records can be updated or deleted, but not lengthened or shortened. In SQL/MP, records (rows) can be updated or deleted, and VARCHAR columns can be lengthened or shortened.

relocate. To move a logically named file from one physical location on disk to another without changing its logical name. Offline relocation does not preserve opens.

rollforward. An HP NonStop Transaction Management Facility (TMF) recovery mechanism that returns a rollforward SQL table, a file, or entire database to its most recent consistent state following a media failure (such as a disk failure). The rollforward facility first uses a previously provided online dump to restore the database to disk and then applies after-images from audit trails to roll the files forward to their most recent consistent state before the failure occurred.

RWEP (read/write/execute/purge). The four types of access to a file on a NonStop system allowed to users of the software. Depending on the security level of the user, the user can examine or copy a file (read); modify the contents of a file (write); execute program code files as a process in TACL (execute); and delete a file, rename it, or alter its definition (purge).

Safeguard. A system-level security tool that provides users of NonStop systems and distributed networks with a set of services for protecting the components of the system or network from unauthorized use. Safeguard services include authentication, authorization, and auditing.

Safeguard command interpreter (SAFECOM). The user interface to Safeguard. SAFECOM commands are used to establish and maintain object authorization records and user authentication records.

storage management. Managing characteristics such as space, performance, availability, creation or destruction, and generations for files and volumes across the storage hierarchy.

storage manager. A process responsible for performing storage management for some set of devices. For example, a pool process is the storage manager for the volumes in its storage pool.

storage pool. A collection of physical volumes. Also known as pool.

structured files. A key-sequenced, relative, or entry-sequenced file. Each record in a structured file contains a set of fields. Data transfers between an application and a structured file are handled in terms of logical records and key fields.

structured query language (SQL). An ANSI-standard and OSI-standard relational database language used to define, manipulate, and control databases. SQL statements can be embedded in programs or entered as commands through the SQLCI for SQL/MP files and MXCI for SQL/MX files. SQL is the standard language for relational database management systems.

structured query language conversational interface (SQLCI). A line-oriented terminal interface that enables a user to enter SQL commands, format and run reports, and operate database utilities.

structured query language (SQL) object program. The object file containing the set of executable machine language instructions produced from a host language source program with embedded SQL statements.

Subsystem Programmatic Interface (SPI). A common, message-based interface that can be used to build and decode messages used for communication between requesters and servers (such as in a management application). It includes procedures to build and decode specially formatted messages; definition files in C, TAL, COBOL, and TACL for inclusion in programs, macros, and routines using the interface procedures; and definition files in DDL for programmers writing their own subsystems.

subvolume. A part of a fully qualified file name. A subvolume is a named logical area on a disk where users can store files that are usually related to one another. Any user can create a subvolume by creating a file and naming the new subvolume as part of the file specifier. A subvolume ceases to exist when the last file in the subvolume is purged. A subvolume name consists of one to eight alphanumeric characters, the first of which must be alphabetic.

super ID. A privileged user who can read, write, execute, and purge all files on the system. The super ID is usually a member of a system-supervisor group. The super ID has the user ID 255,255.

timestamp. An identifier that indicates when a file was created or modified.

HP NonStop Transaction Management Facility (TMF). A product that provides transaction protection and database consistency in demanding online transaction processing (OLTP) and decision-support environments. It gives full protection to transactions that access distributed SQL and Enscribe databases, as well as recovery capabilities for transactions, online disk volumes, and entire databases. To furnish this service, TMF manages database transactions, keeps track of database activity through audit trails, and provides database recovery methods.

unstructured file. An Enscribe file type on a NonStop system that is essentially a byte array on disk that starts at byte address 0 and continues sequentially upward through whatever byte address is identified by the end of file (EOF).

VDP. Abbreviation for virtual disk process.

virtual disk process (VDP). The SMF process that implements location-independent naming. This is a process pair that is treated like a physical disk process from an application viewpoint. Also known as virtual volume process.

virtual volume. A process name used instead of a physical volume name to provide location-independent naming.

virtual volume process. The SMF process managing a logical volume. This is a process pair. Also known as virtual disk process.

volume. A physical storage device (disk) for files on a NonStop system. Volume names always begin with a dollar sign ($). It is the part of the designation that identifies where users store a document. The volume (like a file cabinet) holds subvolumes (like file drawers) that contain files (like individual folders).

$ZSMS. The overall SMF management process; a process pair.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Special Characters